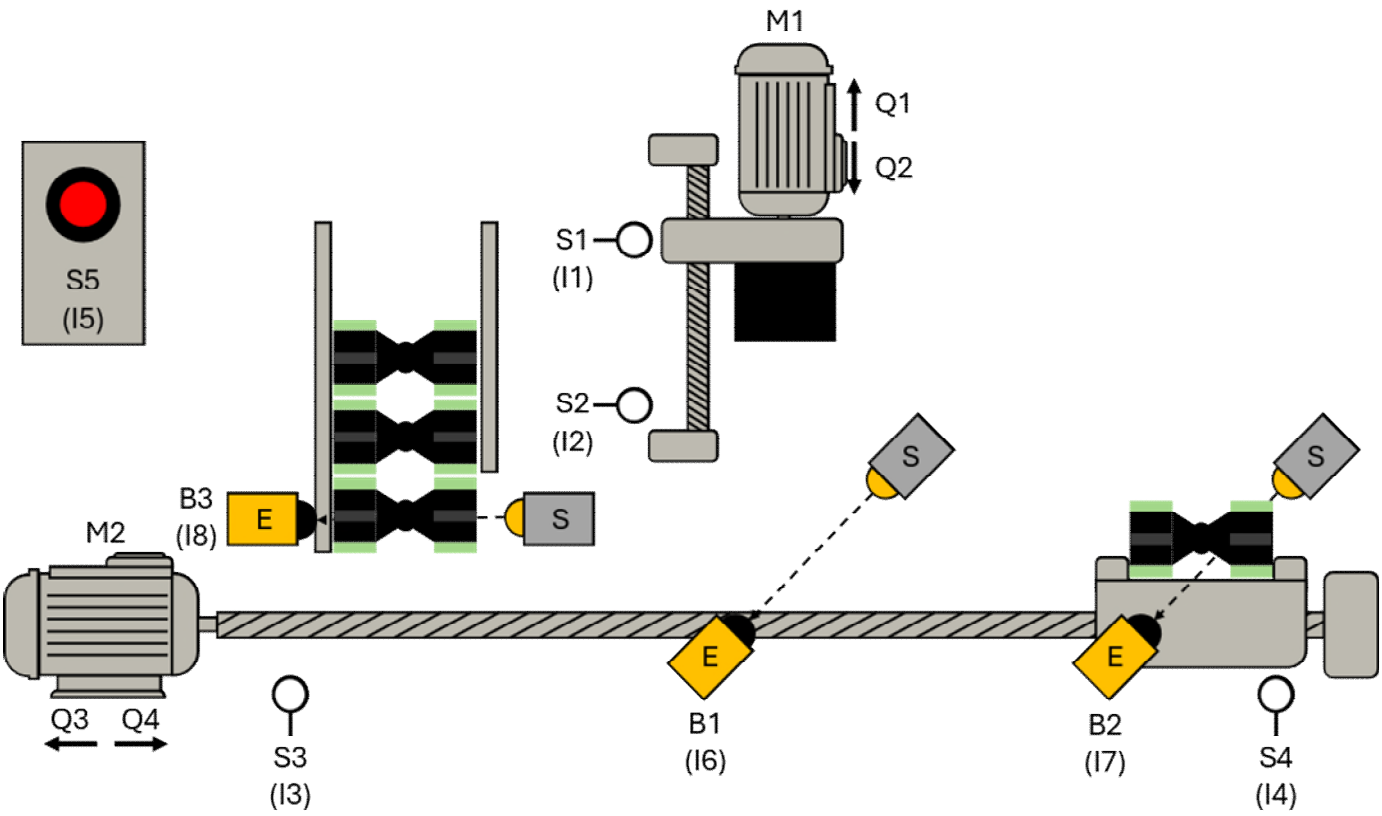
fischertechnik

GROLLMUS

KNOWLEDGE | SOFTWARE | AUTOMATION

Prensa Dobladora 24V

Instrucciones del programa



Índice

7 Instrucciones del programa 1
7.1 Flip-flops 1
7.1.1 Flip-flop de reinicio dominante 3
7.1.2 Flip-flop de ajuste dominante 6
7.1.3 Ejercicio - Comportamiento de dominación 9
7.2 Flancos 14
7.2.1 Reconocer flanco positivo R_TRIG 15
7.2.2 Reconocer flanco negativo F_TRIG 17
7.3 Tiempos 19
7.3.1 Retardo a la conexión TON 21
7.3.2 Retardo de desconexión TOF 23
7.3.3 Pulso TP 25
7.3.4 Ejercicio - Funciones de tiempo IEC 27
7.4 Contador 33
7.4.1 Recuento hacia delante CTU 35
7.4.2 Contando hacia atrás CTD 37
7.4.3 Contar hacia delante y hacia atrás CTUD 39
7.4.4 Ejercicio - Contador IEC 41
7.5 Declaración IF [ST / SCL] 47
7.5.1 IF...THEN - Instrucción 47
7.5.2 Declaración IF...THEN...ELSE 48
7.5.3 IF...THEN...ELSIF - Instrucción 49
7.6 Estructura CASE [ST / SCL] 51

7 Instrucciones del programa

7.1 Chanclas

Un flip-flop es un elemento básico de almacenamiento digital que puede almacenar un estado binario (0 ó 1). Mantiene el estado de una señal momentánea y puede activarse o desactivarse mediante comandos de control especiales.

Las funciones de memoria desempeñan un papel decisivo en la programación de PLC, especialmente en el control de procesos secuenciales y la gestión de estados. Son fundamentales para la implementación de lógicas que van más allá de los simples controles on/off.

Estos son algunos aspectos clave para los que se utilizan las funciones de memoria:

Mantenimiento del estado:

Las funciones de memoria permiten a un PLC mantener el estado de las entradas, los estados internos o las salidas a lo largo del tiempo. Esto es especialmente importante en aplicaciones en las que los estados deben mantenerse a lo largo de un ciclo o incluso de varios ciclos, como ocurre con los procesos de arranque/parada de motores o la supervisión de funciones de seguridad.

Control de procesos:

En los procesos de producción en los que determinados pasos deben realizarse en una secuencia establecida, las funciones de memoria ayudan a guardar el paso actual y activar el siguiente en función de las condiciones cumplidas.

Desmontaje de las señales de entrada:

Las funciones de memoria pueden utilizarse para minimizar los efectos de los rebotes o las interferencias en las señales de entrada. Almacenar un estado estable de una entrada evita que las fluctuaciones a corto plazo provoquen acciones no deseadas.

Tratamiento de interferencias:

Las funciones de memoria permiten registrar y guardar los estados de error en cuanto se producen. El fallo puede restablecerse, por ejemplo, mediante un acuse de recibo del usuario.

Control del proceso:

Las funciones de almacenamiento son indispensables para los procesos que no se ejecutan de forma continua, pero en los que hay que iniciar o detener acciones en función de determinados acontecimientos. Permiten almacenar eventos o estados que pueden recuperarse y procesarse en un momento posterior.

Una función de guardado se caracteriza por un comando de ajuste y un comando de reinicio.
marcado. Encima de la función de memoria debe especificarse un área de datos para guardar el estado de la señal.
La función de memoria puede utilizarse con ajuste dominante (prioridad) o dominante

ejecutarse con un comportamiento de memoria de reinicio. La entrada dominante se identifica con un "1".

7.1.1 Flip-flop de reinicio dominante

Utilice la instrucción para establecer o restablecer el bit de un operando especificado, dependiendo del estado de la señal en las entradas Set y Reset1. El estado actual de la señal del operando se transfiere a la salida Q y puede consultarse en esta salida. El comportamiento dominante se indica mediante el "1" en la entrada Reset.

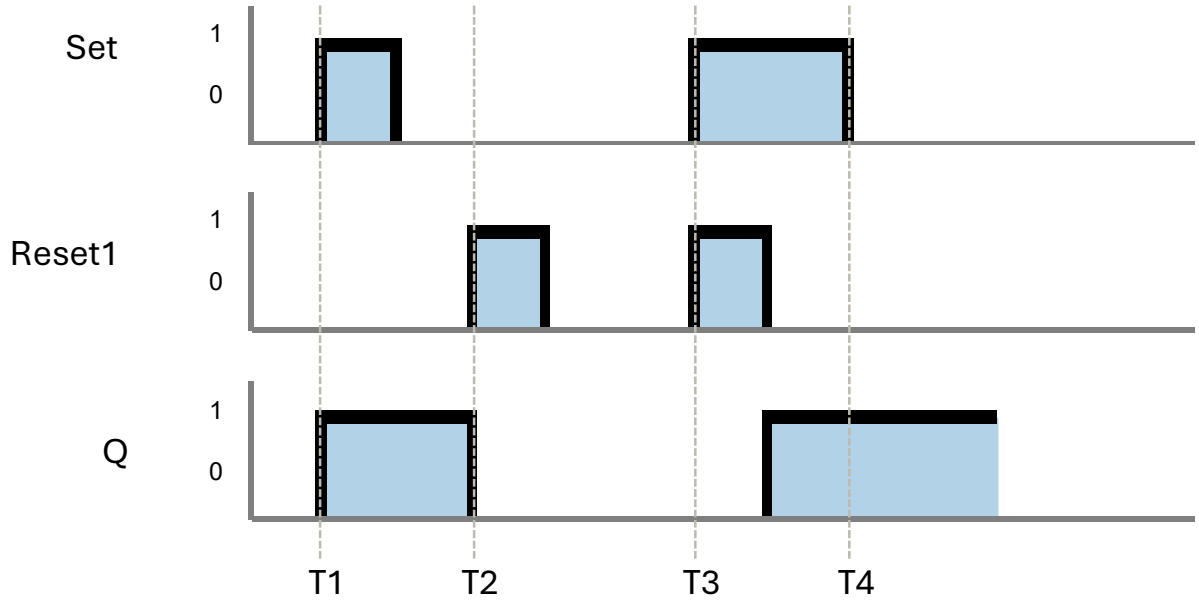


Imagen 1 Diagrama de impulsos - reinicio dominante del flip-flop

T1
Si el estado de la señal en la entrada Set es "1" y en la entrada Reset1 es "0", el operando especificado se pone a "1".
T2
Si el estado de la señal en la entrada Set es "0" y en la entrada Reset1 es "1", el operando especificado se restablece a "0".
T3
La entrada Reset1 domina a la entrada Set. Con un estado de señal de "1" en ambas entradas Set y Reset1, el estado de señal del operando especificado se restablece a "0".
T4
Si el estado de la señal es "0" en ambas entradas Set y Reset1, la instrucción no se ejecuta. En este caso, el estado de señal del operando permanece inalterado.

En Siemens, el flip-flop de reinicio dominante corresponde a la instrucción "SR". Por encima de la llamada al bloque de funciones debe conectarse como memoria una variable del tipo de datos "BOOL".

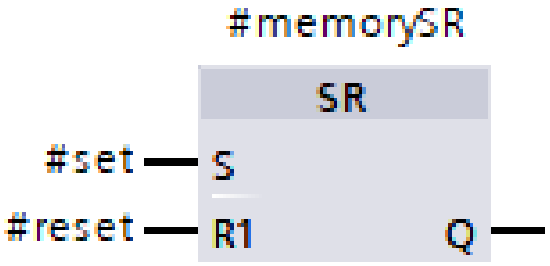


Imagen 2 Reinicialización dominante de flip-flop - Siemens

En CODESYS y Beckhoff, esto se realiza mediante la instrucción "RS". Una instancia del tipo de datos "RS" debe estar conectado por encima de la llamada de bloque de función, ya que se trata de una llamada de bloque de función.

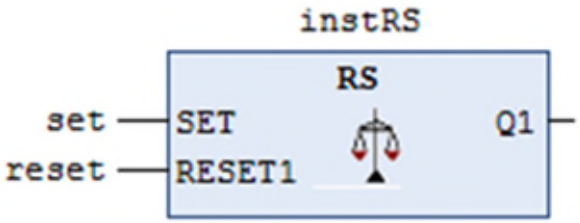


Imagen 3 Reinicio dominante de flip-flop - Beckhoff

En el lenguaje de programación textual ST, los fabricantes de controladores como Beckhoff o CODESYS ofrecen la posibilidad de llamar al elemento RS como flip-flop con dominio de reset, también como bloque de funciones. Para ello, al igual que en FBD, debe declararse una instancia del tipo "RS" en la interfaz del bloque de funciones, que se llama a continuación en la parte de implementación.

```
//Declaration, Interface
VAR
   instRS : RS;   //Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instRS(SET := varBoolSet ,
       RESET1 := varBoolReset,
       Q1 => varBoolQ);
```

Imagen 4 Instrucción ST - reinicio dominante

Esta opción no está disponible con los controladores Siemens. En este caso, el flip-flop debe programarse mediante una instrucción IF.

```
// IF statement
IF Reset1 THEN
   Q := false;
ELSIF Set THEN
   Q := true;
END_IF;
```

Imagen 5 Sentencia IF - reinicio dominante

La función de la instrucción IF se describe detalladamente en este capítulo en "7.5 Instrucción IF [ST /SCL]".

Alternativamente, esto también se puede implementar mediante la vinculación lógica de una auto-retención.

```
//Logical connection
Q := NOT Reset1 AND (Q OR Set);
```

Imagen 6 Conexión lógica - reinicio dominante

7.1.2 Flip-flop de ajuste dominante

Utilice la instrucción Reset o Set para establecer el bit de un operando especificado, dependiendo del estado de la señal en las entradas Reset y Set1. El estado actual de la señal del operando se transfiere a la salida Q y puede consultarse en esta salida.
El comportamiento de dominio se indica mediante el "1" en la entrada de ajuste.

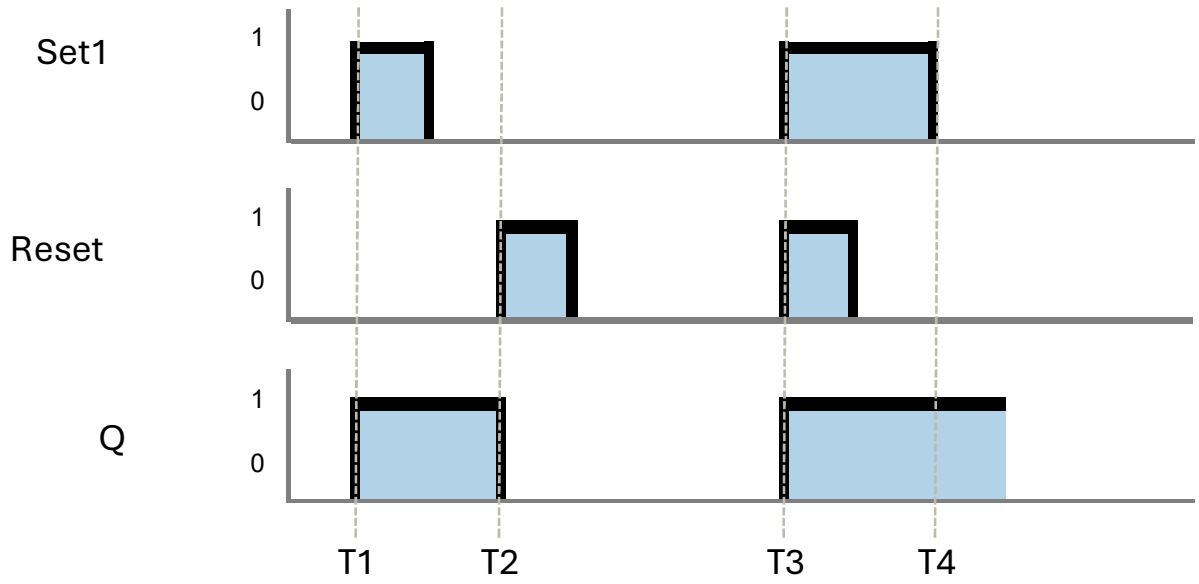


Imagen 7 Diagrama de impulsos - ajuste dominante del flip-flop

T1
Si el estado de la señal en la entrada Set1 es "1" y en la entrada Reset "0", el operando especificado se pone a "1".
T2
Si el estado de la señal en la entrada Reset es "1" y en la entrada Set1 es "0", el operando especificado se restablece a "0".
T3
La entrada Set1 domina a la entrada Reset. Con un estado de señal de "1" en ambas entradas, Reset y Set1, el estado de señal del operando especificado se establece en "1".
T4
Si el estado de la señal es "0" en ambas entradas Reset y Set1, la instrucción no se ejecuta. En este caso, el estado de señal del operando permanece inalterado.

En Siemens, el flip-flop de ajuste dominante corresponde a la instrucción "RS". Por encima de la llamada al bloque de funciones debe conectarse como memoria una variable del tipo de datos "BOOL".

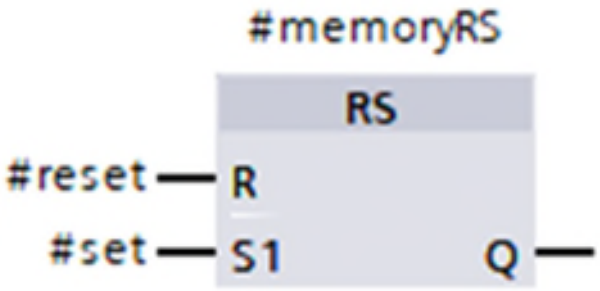


Imagen 8 Ajuste dominante del flip-flop - Siemens

En CODESYS y Beckhoff, esto se realiza mediante la instrucción "SR". Una instancia del tipo de datos "SR" debe estar conectado por encima de la llamada de bloque de función, ya que se trata de una llamada de bloque de función.

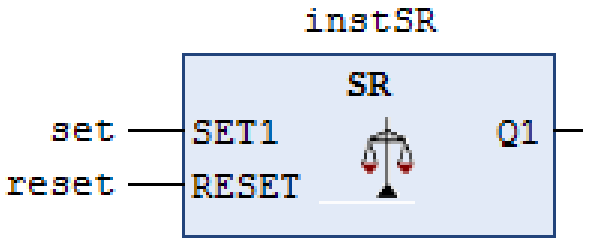


Imagen 9 Ajuste dominante del flip-flop - Beckhoff

En el lenguaje de programación textual ST, los fabricantes de controles como Beckhoff o CODESYS ofrecen la posibilidad de llamar al elemento SR como flip-flop set-dominant, también como bloque de función. Para ello, al igual que en FBD, debe declararse una instancia del tipo "SR" en la interfaz del bloque de funciones, que se llama a continuación en la parte de implementación.

```
//Declaration, Interface
VAR
  instSR : SR;   //Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instSR(SET1 := varBoolSet,
       RESET := varBoolReset,
       Q1 => varBoolQ);
```

Imagen 10 Instrucción ST - ajuste dominante

Esta opción no está disponible con los controladores Siemens. En este caso, el flip-flop debe programarse mediante una instrucción IF.

```
//IF statement
IF Set1 THEN
   Q := true;
ELSIF Reset THEN
   Q := false;
END_IF;
```

Imagen 11 Declaración IF - configuración dominante

La función de la instrucción IF se describe detalladamente en este capítulo en "7.5 Instrucción IF [ST /SCL]".

Alternativamente, esto también se puede implementar mediante la vinculación lógica de una auto-retención.

```
//Logical connection
Q := (NOT Reset AND Q) OR Set1;
```

Imagen 12 Enlace lógico - configuración dominante

7.1.3 Ejercicio - Comportamiento de dominación

Objetivo

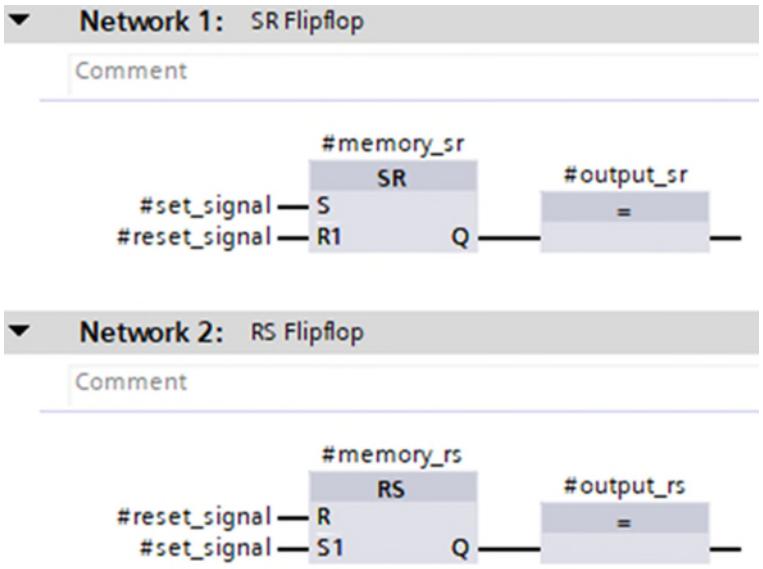
En este ejercicio, el comportamiento de dominancia de los dos flip-flops SR y RS debe programarse en un bloque de función de prueba y aprenderse en la práctica.
El parámetro de entrada "set_signal" se utiliza para controlar la entrada de ajuste de los dos elementos de memoria. El parámetro "reset_signal" se encarga del reset. El estado del flip-flop SR se visualiza a través del parámetro de salida "output_sr", el del flip-flop RS a través de la salida "output_rs".

Tarea

- Cree un bloque de funciones con la siguiente interfaz de bloque de funciones:

		Name	Data type	Comment
1		Input		
2		set_signal	Bool	set
3		reset_signal	Bool	Reset
4		Output		
5		output_sr	Bool	Output SR
6		output_rs	Bool	Output RS
7		InOut		
8		<Add new>		
9		Static		
10		memory_sr	Bool	Internal Memory SR
11		memory_rs	Bool	Internal Memory RS
12		Temp		
13		<Add new>		
14		Constant		
15		<Add new>		

- Se aplicará el siguiente programa:



- Llame el módulo creado en "MAIN".

- Las variables de entrada pueden controlarse y las de salida supervisarse a través de la instancia. Alternativamente, si están disponibles, también pueden conectarse a botones y elementos de visualización a través de la interfaz del bloque de funciones cuando se llama.

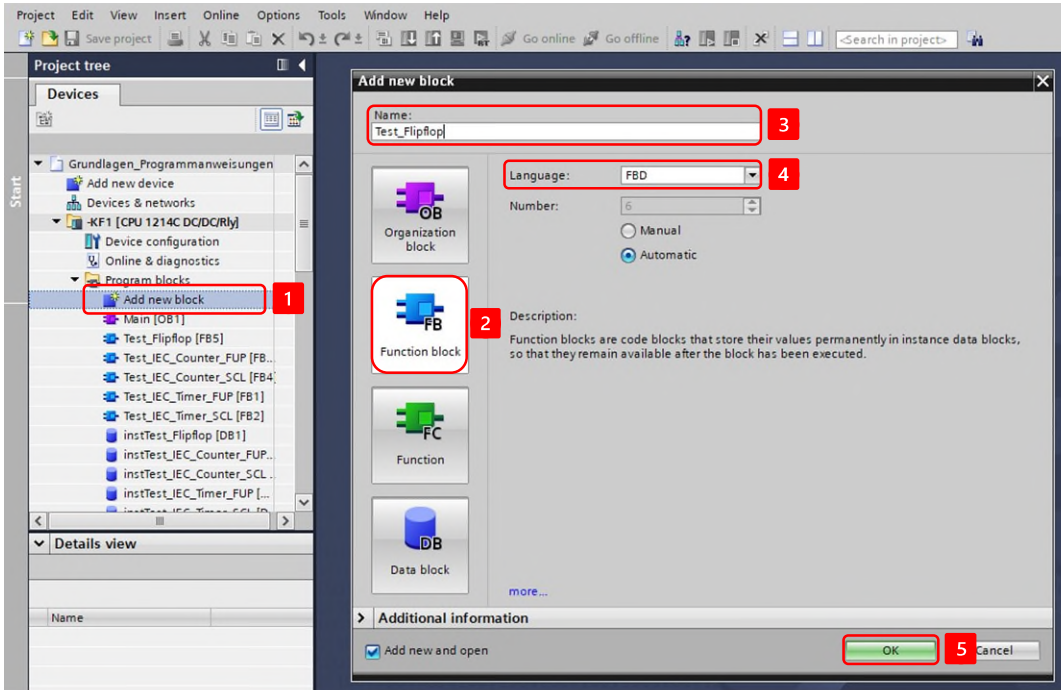
  Si sólo se activa la entrada "set_signal" o "reset_signal", ambos elementos de memoria se comportan de forma idéntica. Si "set_signal" y "reset_signal" se ponen a "TRUE" al mismo tiempo, las salidas "output_sr" y "output_rs" difieren en su estado de señal debido al diferente comportamiento de dominancia.

Como alternativa, también se puede utilizar el bloque preparado "Test_Flipflop [FB5]" del proyecto de plantilla "Grundlagen_Programmanweisungen.zap17".

El capítulo "Puesta en servicio (software)" puede proporcionar ayuda adicional para interpretar el estado del programa.

Procedimiento:

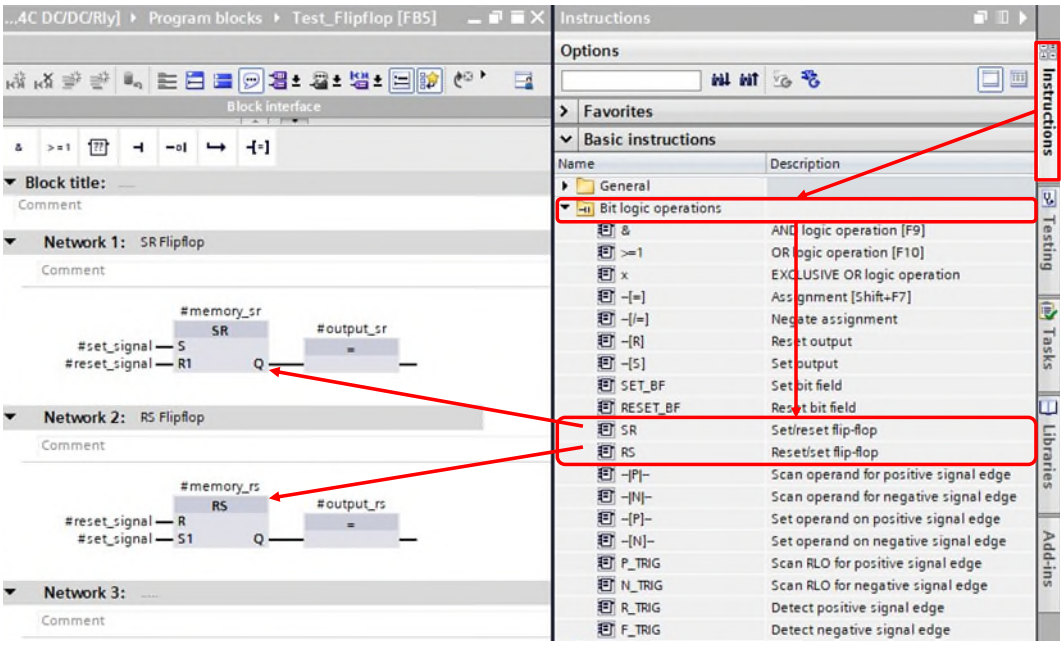
1. cree un nuevo bloque de funciones, seleccione el lenguaje de programación deseado y asígnele un nombre significativo:



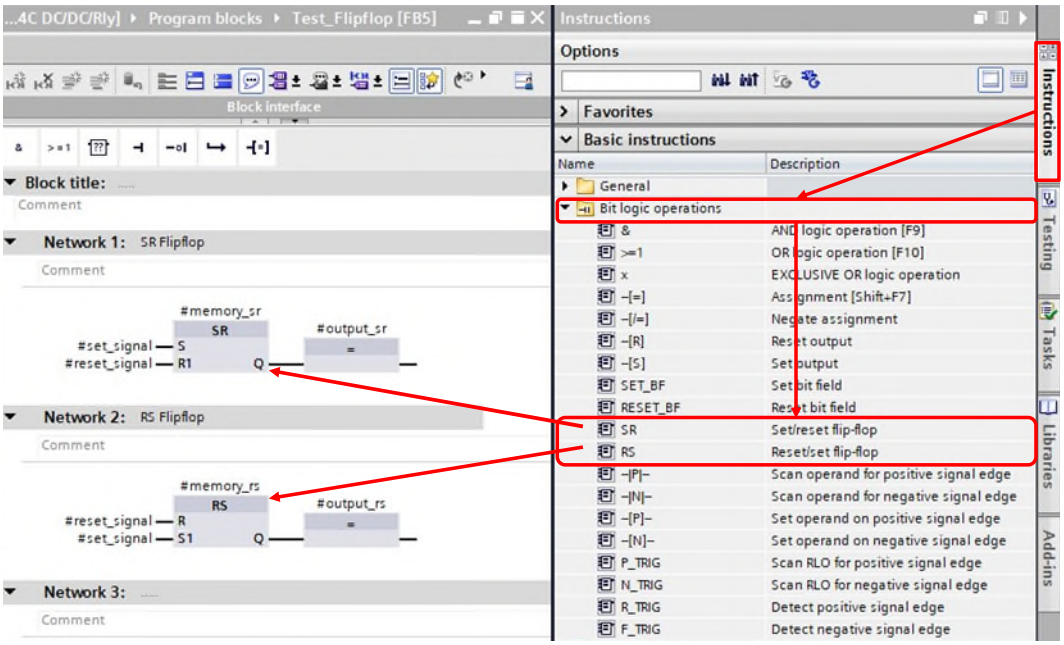
2. declare las siguientes variables en la interfaz del bloque de funciones:

		Name	Data type	Comment
1		Input		
2		set_signal	Bool	set
3		reset_signal	Bool	Reset
4		Output		
5		output_sr	Bool	Output SR
6		output_rs	Bool	Output RS
7		InOut		
8		<Add new>		
9		Static		
10		memory_sr	Bool	Internal Memory SR
11		memory_rs	Bool	Internal Memory RS
12		Temp		
13		<Add new>		
14		Constant		
15		<Add new>		

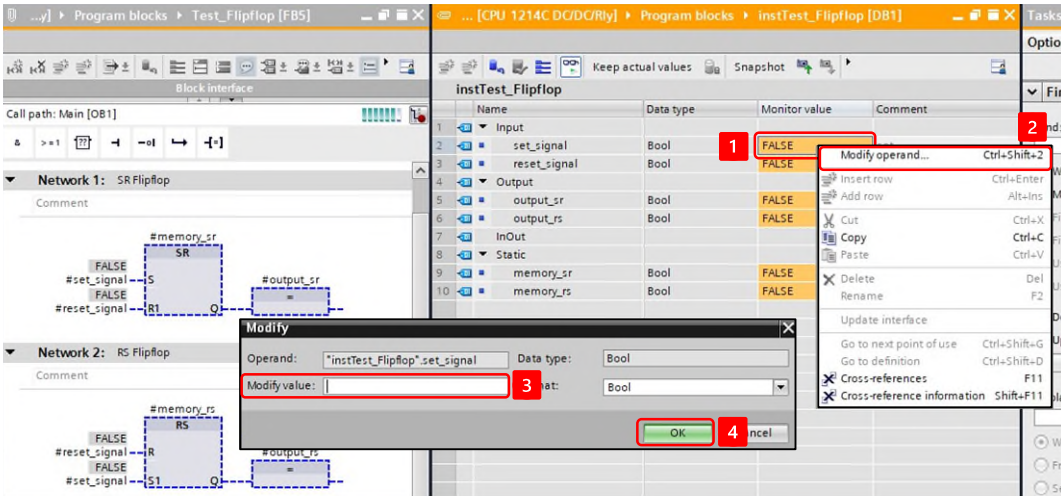
3. implemente el siguiente programa, las funciones de almacenamiento se encuentran en la TaskCard en "Instrucciones"→ "Instrucciones simples"→ "Enlaces de bits" . Éstos pueden arrastrarse y soltarse en el área de trabajo:

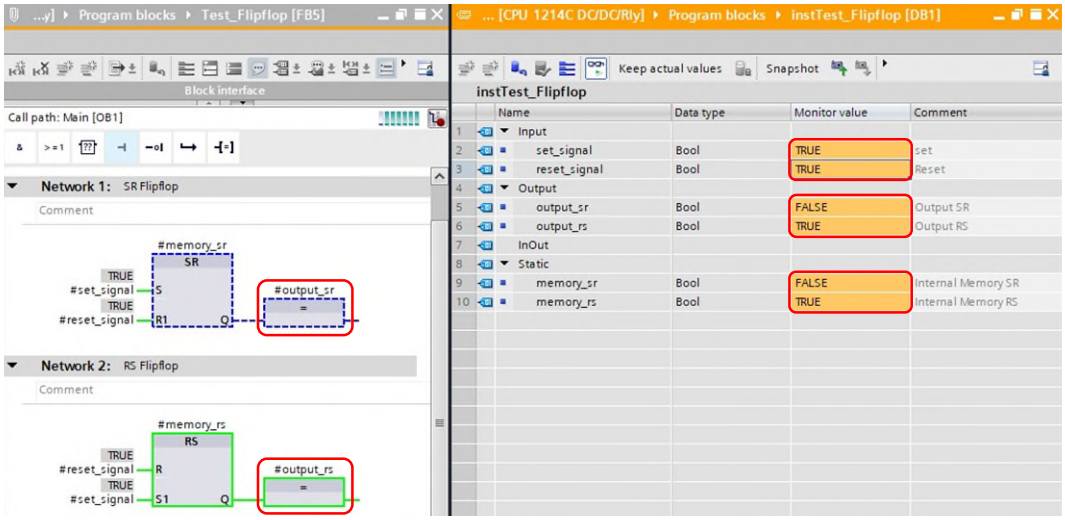


4. llame al bloque de funciones en "MAIN" y cree una instancia.



5. Utilice la instancia para ajustar las variables de entrada "set_signal" y "reset_signal" a "TRUE" una tras otra, luego ajuste ambas variables de entrada a "TRUE" al mismo tiempo y controle las variables de salida "output_sr" y "output_rs":





7.2 Flancos

Las aristas en la programación de PLC son importantes para reconocer ciertos eventos que ocurren cuando la señal cambia. Una evaluación de flanco registra si el estado de una señal binaria ha cambiado en comparación con el ciclo de programa anterior.

Según la norma IEC 61131, existen los siguientes bordes:

- Flanco positivo (R_TRIG)
- Flanco negativo (F_TRIG)

Los flancos son necesarios, por ejemplo, para:

Detección de eventos:

Los bordes permiten reconocer eventos específicos que se producen exactamente cuando se produce la transición de una señal. Esto es útil para activar acciones exactamente cuando cambia una señal, no mientras permanece en un estado determinado.

Procesos controlados por reloj:

En muchas aplicaciones, es necesario iniciar procesos de forma sincronizada con determinados cambios de señal.

Acciones controladas por eventos:

En la tecnología de automatización, a menudo es necesario iniciar determinadas acciones en el momento exacto en que se acciona un interruptor o un sensor detecta un evento. La detección de bordes lo hace posible con precisión y evita fallos de funcionamiento.

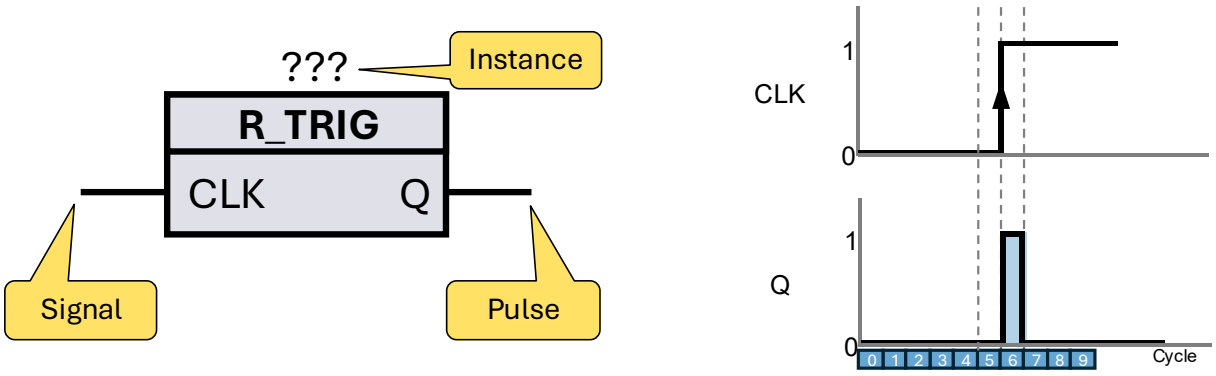
Interrupciones y tiempos:

En las aplicaciones en tiempo real, los bordes pueden utilizarse para generar interrupciones con el fin de reaccionar inmediatamente a eventos externos. También permiten cronometrar y medir el tiempo con precisión.

Cuando cambia la señal (flanco) de una señal binaria, la evaluación del flanco emite un impulso. Este impulso sólo está presente durante un ciclo.
Para detectar el cambio de señal (flanco), se compara el estado actual (estado real) con el estado anterior (valor pasado) de la señal. En cada ciclo de programa, el estado anterior de la señal (valor histórico) se compara con el estado actual de la señal. Para ello, la evaluación requiere una memoria (instancia).

7.2.1 Reconocer flanco positivo R_TRIG

Con la instrucción IEC "R_TRIG", detectar flanco positivo de señal, puede detectar un cambio de estado de "0" a "1" en la entrada CLK.



Fotografía 13 Instrucción FBD R_TRIG y diagrama de impulsos

Ciclo actual 6
La instrucción compara el valor actual en la entrada CLK (ciclo 6) con el estado de la consulta anterior (memoria de flanco, ciclo 5), que se almacena en la instancia.

- Si CLK es "1" y la memoria de flanco es "0", se ha detectado un flanco positivo.
- Si la instrucción ha detectado un cambio de estado en la entrada CLK de "0" a "1", la salida Q se pone a "1".
- La memoria de flancos se ajusta al estado de la señal actual. Memoria de flanco "1".

Ciclo actual 7
La instrucción compara el valor actual en la entrada CLK (ciclo 7) con el estado de la consulta anterior (memoria de flanco, ciclo 6), que se almacena en la instancia.

- Si CLK es "1" y la memoria de flanco es "1", no se ha detectado ningún flanco.
- Si la instrucción no ha detectado un cambio de señal en la entrada CLK de "0" a "1", la salida Q se pone a "0".
- La memoria de flancos se ajusta al estado de la señal actual. Memoria de flanco "1".

Si la instrucción detecta un cambio de señal en la entrada CLK de "0" a "1", se genera un impulso en la salida Q, es decir, la salida lleva el valor "1" durante un ciclo. En todos los demás casos, el estado de la señal a la salida de la instrucción es "0".

A cada llamada de la instrucción "Detectar flanco positivo de señal" se le debe asignar una instancia del tipo de datos "R_TRIG", en la que se almacenan los datos de la instancia.

Conversión textual Reconocer el flanco positivo de la señal (R_TRIG)
El módulo de bordes R_TRIG se instanciará de forma similar a un módulo de funciones. Se necesita una zona de memoria independiente para cada llamada del bloque de funciones "R_TRIG". Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

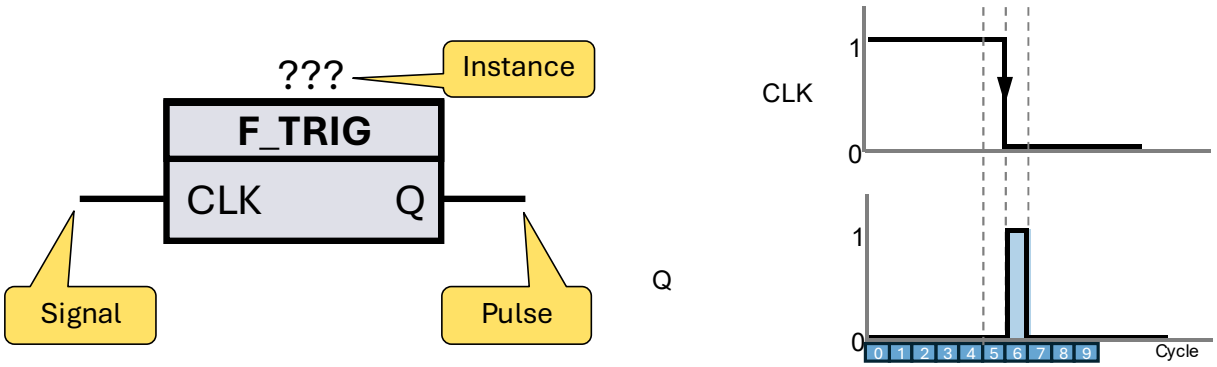
Ejemplo:

```
//Declaration, Interface
VAR
   instRtrig : R_TRIG;   // Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instRtrig(CLK := varBoolClk,
          Q => varBoolQ);
```

Imagen 14 Instrucción ST/SCL R_TRIG

7.2.2 Reconocer flanco negativo F_TRIG

Puede utilizar la instrucción "Detectar flanco negativo de señal" para detectar un cambio de estado de "1" a "0" en la entrada CLK.



Fotografía 15 Instrucción FBD F_TRIG y diagrama de impulsos

Ciclo actual 6
La instrucción compara el valor actual en la entrada CLK (ciclo 6) con el estado de la consulta anterior (memoria de flanco, ciclo 5), que se almacena en la instancia.

- Si CLK es "0" y la memoria de flanco es "1", se ha detectado un flanco negativo.
- Si la instrucción ha detectado un cambio de señal en la entrada CLK de "1" a "0", la salida Q se pone a "1".
- La memoria de flancos se ajusta al estado de la señal actual. Memoria de flanco "0".

Ciclo actual 7
La instrucción compara el valor actual en la entrada CLK (ciclo 7) con el estado de la consulta anterior (memoria de flanco, ciclo 6), que se almacena en la instancia.

- Si CLK es "0" y la memoria de flancos es "0", no se ha detectado ningún flanco.
- Si la instrucción no ha detectado un cambio de señal en la entrada CLK de "1" a "0", la salida Q se pone a "0".
- La memoria de flancos se ajusta al estado de la señal actual. Memoria de flanco "0".

Si la instrucción detecta un cambio de estado en la entrada CLK de "1" a "0", se genera un impulso en la salida Q, es decir, la salida lleva el valor "1" durante un ciclo. En todos los demás casos, el estado de la señal a la salida de la instrucción es "0".

A cada llamada de la instrucción "Detectar flanco negativo de señal" se le debe asignar una instancia del tipo de datos "F_TRIG", en la que se almacenan los datos de la instancia.

Conversión textual Reconocer el flanco negativo de la señal (F_TRIG)
El módulo de bordes F_TRIG se instanciará de forma similar a un módulo de funciones. Se requiere una zona de memoria independiente para cada llamada del bloque de funciones "F_TRIG". Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

Ejemplo:

```
//Declaration, Interface
VAR
   instFtrig : F_TRIG;   // Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instFtrig(CLK := varBoolClk,
          Q => varBoolQ);
```

Imagen 16 Instrucción ST/SCL F_TRIG

Comportamiento tras el arranque de la CPU
La norma IEC 61131 describe que la instrucción "F_TRIG" pone la salida "Q" a TRUE durante un ciclo si la entrada "CLK" tiene el valor FALSE al arrancar la CPU.

7.3 Times

Las funciones de tiempo pueden utilizarse para lanzar acciones controladas por tiempo en el programa. Se activan mediante una señal binaria. El resultado de la evaluación también es una señal binaria. Las funciones temporales son instrucciones que tienen una instancia.

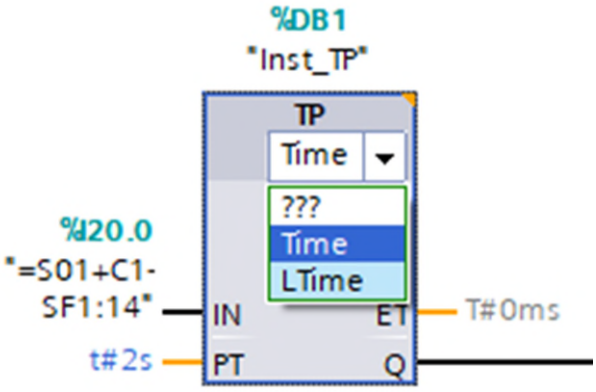
Las siguientes funciones de tiempo están disponibles de acuerdo con IEC 61131:

- Generación de impulsos (TP)
- Retardo a la conexión (TON)
- Retardo a la desconexión (TOF)

El intervalo de tiempo máximo que puede mostrarse depende del tipo de datos seleccionado.

- Formato TIME (32 bits) rango de tiempo máximo: ~24 días
- Formato LTIME (64 bits) rango de tiempo máximo: ~106751 días (292 años)

En el Portal TIA, el tipo de datos puede establecerse directamente en la instrucción mediante el menú desplegable.



Fotografía 17 Tipo de datos de la función de tiempo (Siemens)

En Codesys / Beckhoff, existen bloques de funciones independientes para las operaciones temporales de 64 bits:

- LTP
- LTON
- LTOF

Ejemplos de valores razonables:

Sintaxis	Significado
T#5s	5 segundos
T#1d3h5m30s500ms	1 día, 3 horas, 5 minutos, 30 segundos y 500 milisegundos
LTIME#50d3h	50 días y 3 horas

He aquí algunas aplicaciones importantes y las ventajas de las funciones de tiempo en la programación de PLC:

Retrasos en el control:

Las funciones de tiempo permiten introducir retardos en los procesos de control. Por ejemplo, se puede retrasar el arranque de un motor tras la activación de una señal de disparo.

Control de secuencia:

Si los procesos deben ejecutarse en determinados intervalos de tiempo, las funciones de tiempo garantizan que cada paso se active durante el tiempo definido.
lo hará.

Control de las condiciones temporales:

Las funciones de tiempo ayudan a supervisar los procesos garantizando que determinadas acciones se completen dentro de los plazos establecidos.

Generación de impulsos:

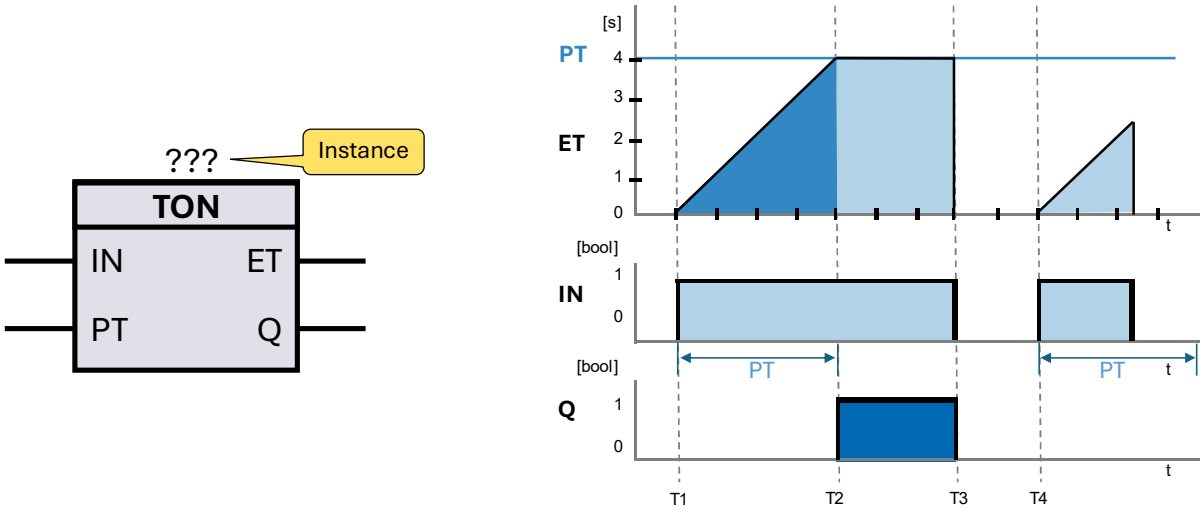
En muchas aplicaciones se necesitan señales pulsantes. Las funciones de tiempo generan estos pulsos a intervalos predefinidos.

Aplicar los tiempos de espera:

Las funciones de tiempo son útiles para establecer tiempos de espera, por ejemplo, para evitar la sobrecarga de la máquina.

7.3.1 Retardo a la conexión TON

La instrucción "Generar retardo a la conexión" retarda el ajuste de la salida Q (Salida) durante el tiempo parametrizado PT (Tiempo prefijado). El valor de tiempo actual puede consultarse en la salida ET (Tiempo transcurrido). El valor de tiempo comienza en T#0s y finaliza cuando se alcanza el tiempo de duración PT. En cuanto el estado de la señal en la entrada IN (Entrada) cambia a "0", la salida ET se pone a cero.



Fotografía 18 Instrucción FBD TON y diagrama de impulsos

T1
La instrucción comienza cuando el resultado de la operación lógica (VKE) en la entrada IN cambia de "0" a "1" (flanco de señal positivo). A partir de este momento comienza a correr el tiempo programado PT.
T2
Una vez transcurrido el tiempo PT, la salida Q pasa al estado de señal "1".
T3
Q permanece activada mientras la entrada de inicio sea "1". Si el estado de la señal en la entrada de inicio cambia de "1" a "0", la salida Q se restablece.
T4
Un nuevo flanco positivo de señal en la entrada de inicio reinicia la función de tiempo.

A cada llamada de la instrucción "Crear retardo a la conexión" se le debe asignar una instancia del tipo de datos "TON", en la que se almacenan los datos de la instancia.

Conversión textual Generar retardo de conexión (TON)
El retardo de conexión TON se instanciará de forma similar a un bloque de funciones. Para cada llamada del bloque de funciones "TON" se necesita una zona de memoria independiente. Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

Ejemplo:

```
//Declaration, Interface
VAR
   instTon : TON;   // Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instTon(IN := varBoolIn,
        PT := varTimePt,
        Q => varBoolQ,
        ET => varTimeEt);
```

Fotografía 19 Instrucción ST/SCL TON

7.3.2 Retardo de desconexión TOF

Utilice la instrucción "Generar retardo de desconexión" para retardar el rearme de la salida Q (Salida) durante el tiempo programado PT (Tiempo prefijado). El valor de tiempo actual puede consultarse en la salida ET (Tiempo transcurrido). El valor de tiempo comienza en T#0s y finaliza cuando se alcanza el tiempo de duración PT. Una vez transcurrido el tiempo PT, la salida ET permanece en el valor actual hasta que la entrada IN vuelve a cambiar a "1". Si la entrada IN cambia a "1" antes de que haya transcurrido el tiempo PT, la salida ET se restablece a T#0s.

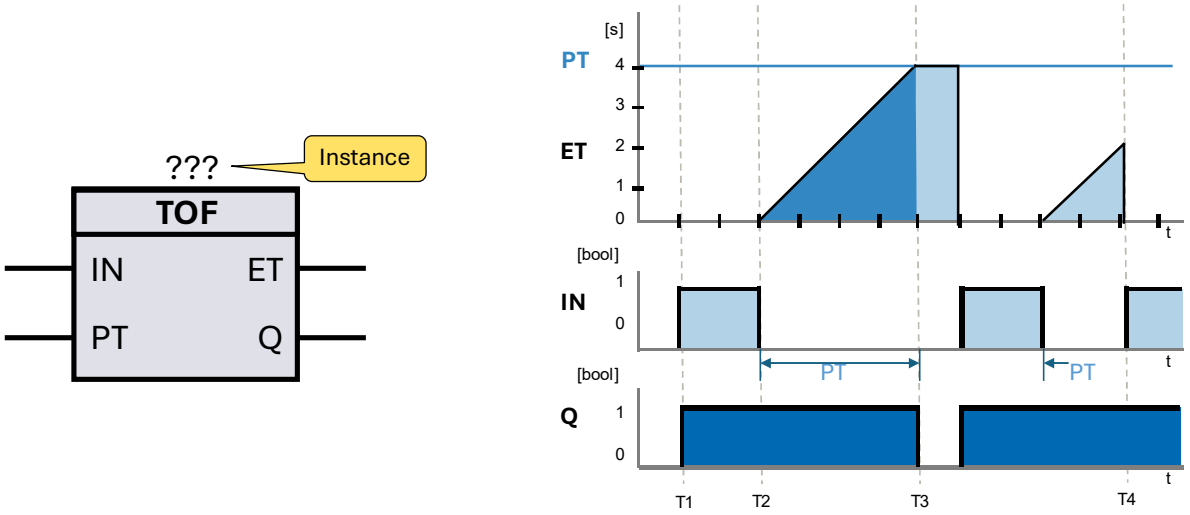


Imagen 20 Instrucción FBD TOF y diagrama de impulsos

T1
La salida Q se activa cuando el resultado de la operación lógica (VKE) en la entrada IN cambia de "0" a "1" (flanco positivo).
T2
Cuando la entrada IN vuelve a "0" (flanco negativo), el tiempo PT programado comienza a correr. La salida Q permanece activada mientras el tiempo PT esté en marcha.
T3
Una vez transcurrido PT, la salida Q se reinicia.
T4
Si la señal de entrada IN cambia a "1" antes de que haya transcurrido el tiempo PT, el tiempo se reinicia y la salida Q permanece activada.

A cada llamada de la instrucción "Crear retardo de desconexión" se le debe asignar una instancia del tipo de datos "TOF", en la que se guardan los datos de la instancia.

Conversión textual Generar retardo de desconexión (TOF)
El retardo de desconexión TOF se instanciará de forma similar a un bloque de funciones. Para cada llamada del bloque de funciones "TOF" se necesita una zona de memoria propia. Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

Por ejemplo:

```
//Declaration, Interface
VAR
  instTof : TOF;   // Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instTof(IN := varBoolIn,
        PT := varTimePt,
        Q => varBoolQ,
        ET => varTimeEt);
```

Imagen 21 Instrucción ST/SCL TOF

7.3.3 Pulso TP

Utilice la instrucción "Generar impulso" para activar la salida Q (Salida) durante un periodo de tiempo programado. Puede consultar el valor de tiempo actual en la salida ET (Tiempo transcurrido). El valor de tiempo comienza en T#0s y finaliza cuando se alcanza el valor de PT (Tiempo prefijado). Si PT ha transcurrido y la señal de entrada IN (Entrada) es "0", la salida ET se pone a cero.

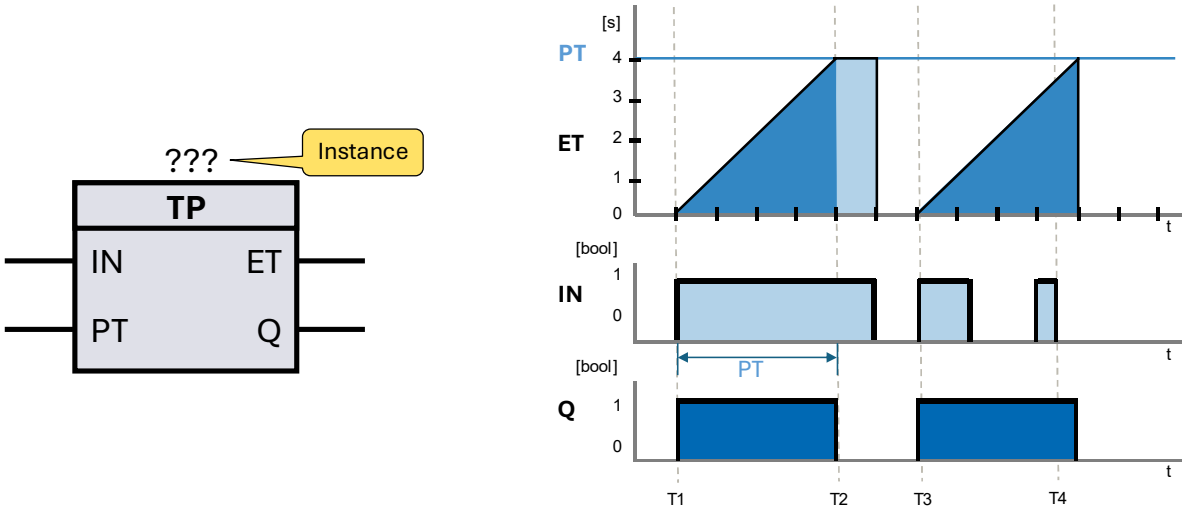


Imagen 22 Instrucción FBD TP y diagrama de impulsos

T1
Cuando la entrada IN cambia de "0" a "1", el tiempo programado PT empieza a correr y la salida Q cambia a "1".
T2
Una vez transcurrido el tiempo PT, la salida Q pasa de "1" a "0", independientemente de que la señal de entrada IN siga siendo "1".
T3
Cuando el periodo de tiempo PT ha transcurrido y la entrada IN cambia de "0" a "1", el periodo de tiempo programado PT empieza a correr de nuevo y la salida Q cambia a "1".
T4
La salida Q permanece activada mientras dura PT, independientemente de cómo siga comportándose la señal de entrada. Mientras PT está en marcha, un nuevo flanco positivo en la entrada IN no influye en la salida Q ni en el tiempo transcurrido PT.

A cada llamada de la instrucción "Generar impulso" se le debe asignar una instancia del tipo de datos "TP" en la que se guardan los datos de la instancia.

Conversión textual Generar impulso (TP)
El bloque de funciones de impulsos TP se instancian de forma similar a un bloque de funciones. Se necesita una zona de memoria independiente para cada llamada del bloque de funciones "TP". Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

Por ejemplo:

```
//Declaration, Interface
VAR
   instTp : TP;   // Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instTp(IN := varBoolIn,
       PT := varTimePt,
       Q => varBoolQ,
       ET => varTimeEt);
```

Fotografía 23 Instrucción ST/SCL TP

7.3.4 Ejercicio - Funciones horarias CEI

Objetivo

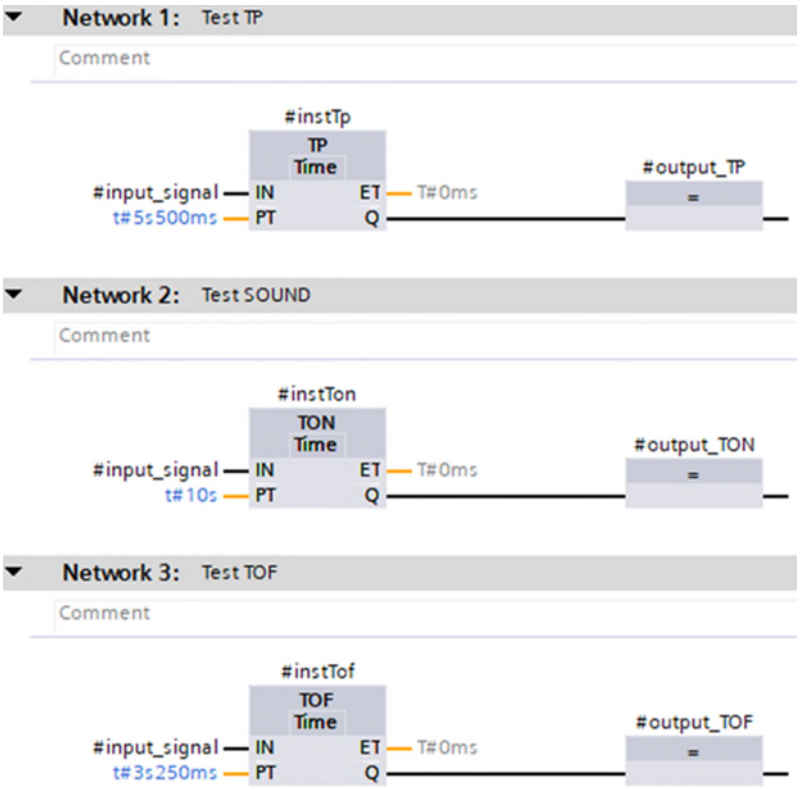
En este ejercicio se programan las 3 funciones de tiempo IEC (TP, TON, TOF) en un bloque de funciones de prueba y se familiariza con ellas en la práctica. Las funciones de tiempo pueden iniciarse mediante el parámetro de entrada "input_signal". El estado se muestra a través de los parámetros de salida "output_TP", "output_TON", "output_TOF".

Tarea

- Cree un bloque de funciones con la siguiente interfaz de bloque de funciones:

	Name	Data type	Comment
1	Input		
2	input_signal	Bool	Test Input
3	Output		
4	output_TP	Bool	Test Output TP
5	output_TON	Bool	Test Output TON
6	output_TOF	Bool	Test Output TOF
7	InOut		
8	<Add new>		
9	Static		
10	instTp	TP_TIME	Instance TP
11	instTon	TON_TIME	Instance TON
12	instTof	TOF_TIME	Instance TOF
13	Temp		

- En función del lenguaje de programación seleccionado, se aplicará el siguiente programa:




```
1  //Test TP
2 #instTp(IN := #input_signal,
3         PT := t#5s500ms,
4         //ET => ,
5         Q => #output_TP);
6
7  //Test TON
8 #instTon(IN := #input_signal,
9          PT := t#10s,
10         //ET => ,
11         Q => #output_TON);
12
13
14 //Test TOF
15 #instTof(IN := #input_signal,
16         PT := t#3s250ms,
17         //ET => ,
18         Q => #output_TOF);
```

- Llame el módulo creado en "MAIN".

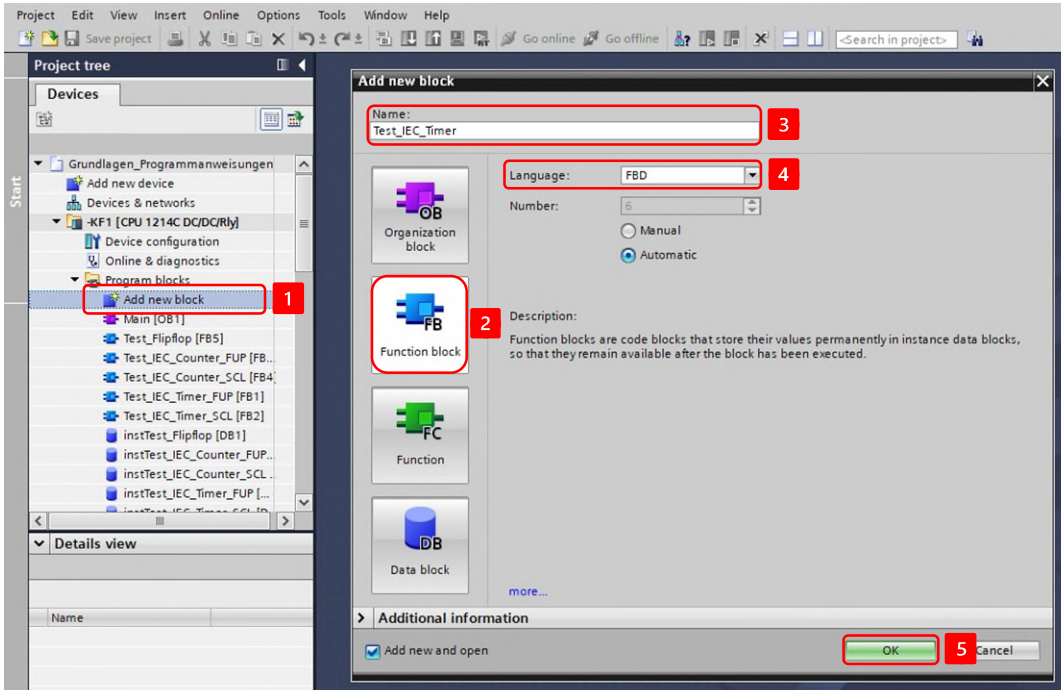
- Las variables de entrada pueden controlarse y las de salida supervisarse a través de la instancia. Alternativamente, si están disponibles, también pueden conectarse a botones y elementos de visualización a través de la interfaz del bloque de funciones cuando se llaman. El paso del tiempo también puede controlarse en las instancias individuales de las funciones de tiempo.

Como alternativa, también se pueden utilizar los bloques preparados "Test_IEC_Timer_FUP[FB1]" o "Test_IEC_Timer_SCL[FB2]" del proyecto de plantilla "Grundlagen_Programmanweisungen.zap17".

El capítulo "Puesta en servicio (software)" puede proporcionar ayuda adicional para interpretar el estado del programa.

Procedimiento:

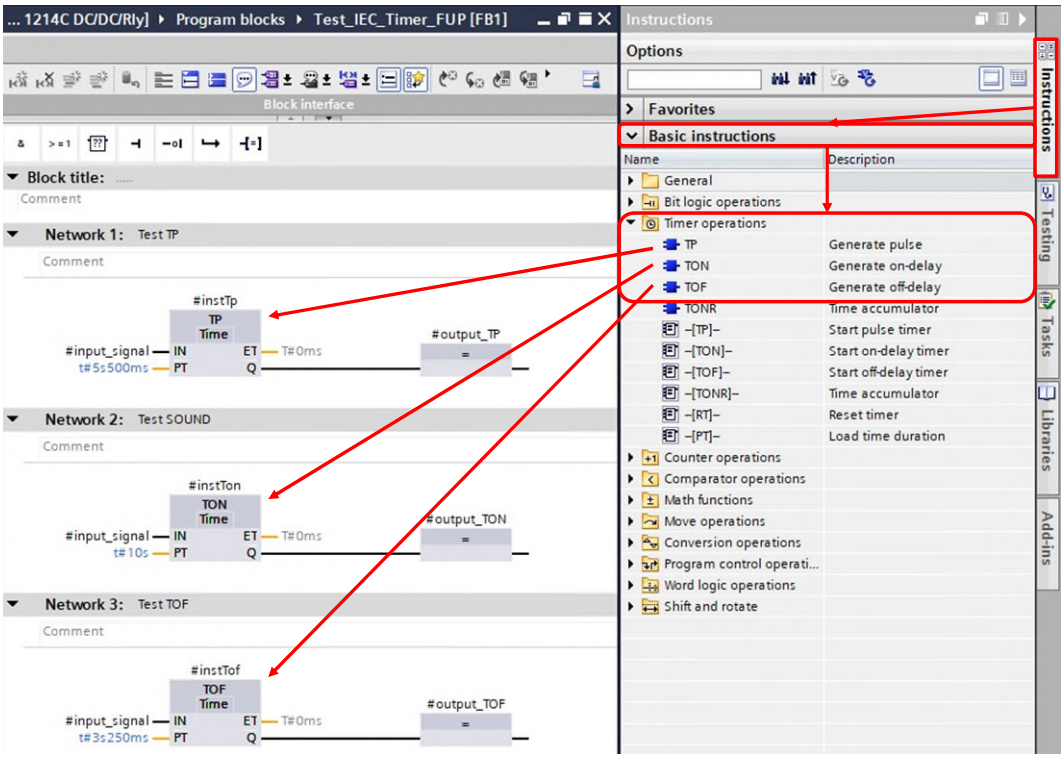
1. cree un nuevo bloque de funciones, seleccione el lenguaje de programación deseado y asígnele un nombre significativo:



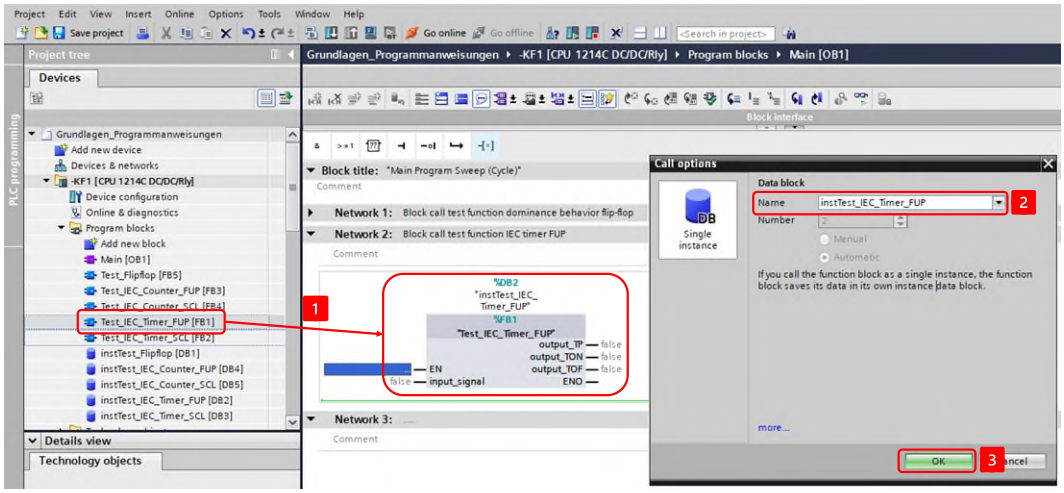
2. declare las siguientes variables en la interfaz del bloque de funciones:

	Name	Data type	Comment
1	Input		
2	input_signal	Bool	Test Input
3	Output		
4	output_TP	Bool	Test Output TP
5	output_TON	Bool	Test Output TON
6	output_TOF	Bool	Test Output TOF
7	InOut		
8	<Add new>		
9	Static		
10	instTp	TP_TIME	Instance TP
11	instTon	TON_TIME	Instance TON
12	instTof	TOF_TIME	Instance TOF
13	Temp		

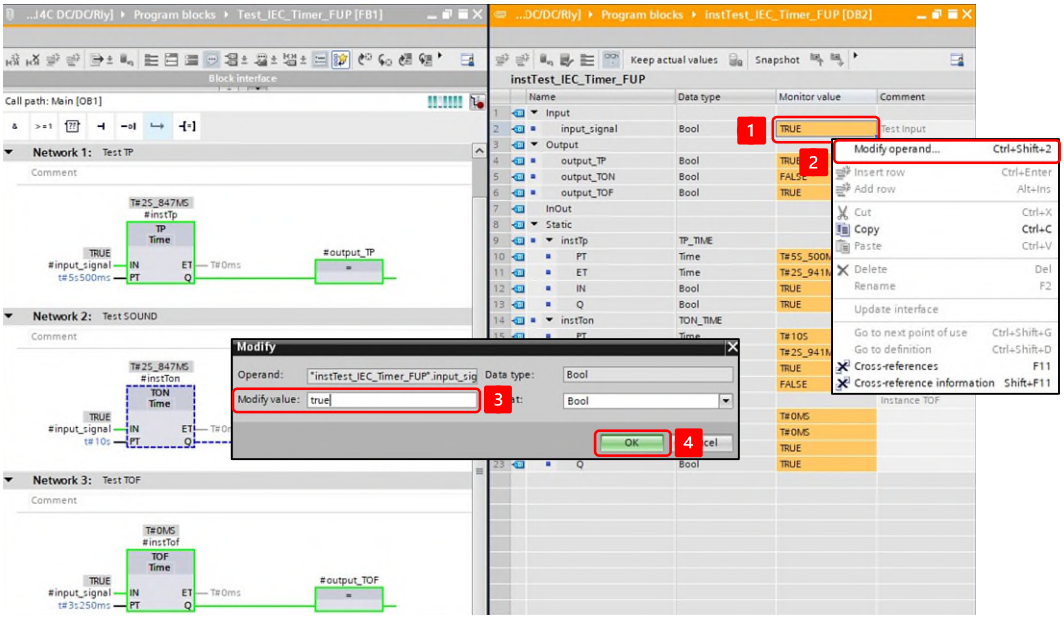
3. implemente el siguiente programa, las funciones de tiempo se encuentran en la TaskCard en "Instrucciones"→ "Instrucciones simples"→ "Tiempos" :

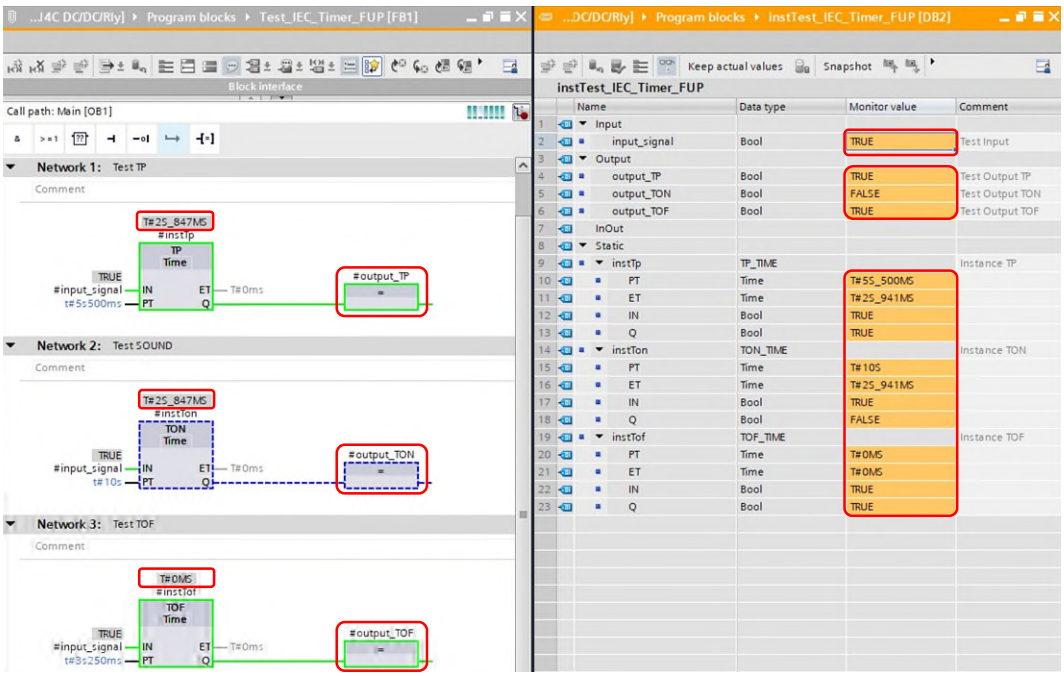


4. llama al módulo de funciones en "MAIN" y crea una instancia:



5. controlar la variable de entrada "input_signal" a través de la instancia y observar las variables de salida "output_TP", "output_TON", "output_TOF", así como las instancias individuales de las funciones temporales en el bloque de datos de instancia correspondiente:





7.4 Contador

Las funciones de conteo pueden contar un valor numérico hacia arriba o hacia abajo en respuesta al pulso de una señal binaria. Se trata de instrucciones que requieren una instancia.

Las siguientes funciones de recuento están disponibles de conformidad con la norma IEC 61131:

- Contador ascendente (CTU)
- Contador descendente (CTD)
- Contador de subidas y bajadas (CTUD)

El rango de recuento máximo visualizable depende del tipo de datos seleccionado.

En el Portal TIA, el tipo de datos puede establecerse directamente en la instrucción mediante el menú desplegable.

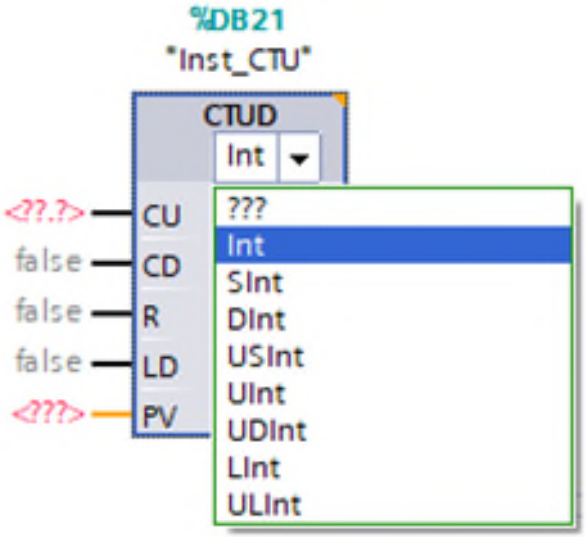


Imagen 24 Tipo de datos de la función de recuento (Siemens)

Codesys dispone de sus propios bloques de funciones para las operaciones de recuento de 64 bits:

- LCTU
- LCTD
- LCTUD

He aquí algunas aplicaciones importantes y las ventajas de las funciones de recuento en el
Tecnología de automatización:

Control de la producción:

Las funciones de recuento se utilizan para controlar el número de unidades producidas. Esto es especialmente importante en la industria manufacturera para garantizar el cumplimiento de los objetivos de producción.

Control de inventarios y flujo de materiales:

Ayudan a controlar los movimientos de material dentro de una instalación de producción. Por ejemplo, es posible contar con qué frecuencia pasa una pieza por una estación determinada, lo que ayuda a optimizar el flujo de materiales y controlar los niveles de existencias.

Aplicaciones de seguridad:

Las funciones de recuento también pueden utilizarse en aplicaciones de seguridad crítica para controlar el número de veces que se ha utilizado un dispositivo o una máquina. Esto puede ser importante para cumplir los intervalos de mantenimiento.

Control de procesos secuenciales:

Las funciones de recuento permiten controlar con precisión la frecuencia con la que se ejecutan determinadas acciones en un programa. Son especialmente útiles en situaciones en las que un proceso debe repetirse varias veces antes de que la secuencia del programa pase al siguiente paso.

Procesamiento por lotes:

En la industria química o en el procesado de alimentos, las funciones de recuento pueden utilizarse para controlar el número de lotes que han pasado por un proceso, lo que contribuye al control de calidad y a la documentación.

7.4.1 Cuenta hacia delante CTU

Puede programar un contador ascendente con el contador IEC-CTU. Un flanco positivo en la entrada CU (Count Up) aumenta el valor en la salida CV (Current Value). Si se alcanza o sobrepasa el valor especificado en la entrada PV (Preset Value), se conmuta la salida Q (Output). El valor del contador puede ponerse a 0 a través de la entrada R (Reset).

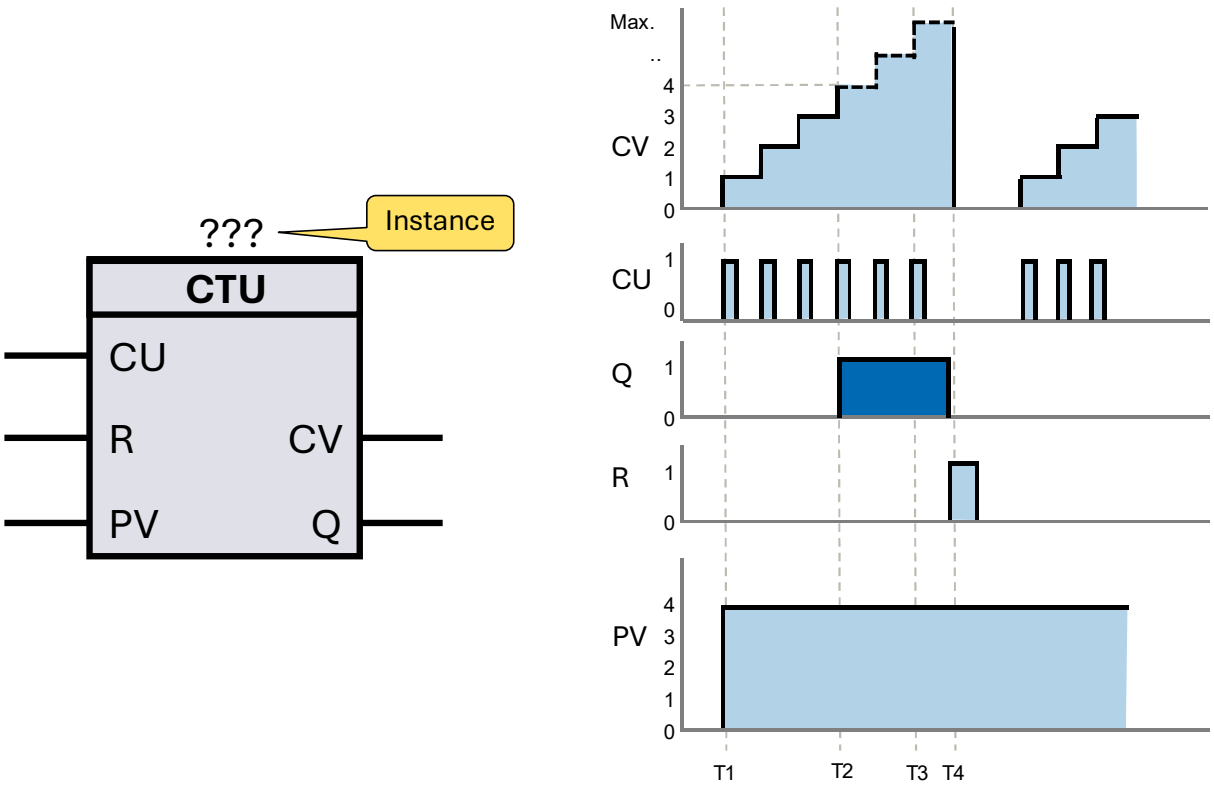


Imagen 25 Instrucción FBD CTU y diagrama de impulsos

T1
Si la señal en la entrada CU cambia de "0" a "1" (flanco positivo), el valor del contador en la salida CV se incrementa en uno.
T2
La salida Q indica el estado del contador. El estado de la señal de Q viene determinado por la consigna PV parametrizada. Si el valor actual del contador alcanza o supera la consigna PV parametrizada, la salida Q se pone a "1". En caso contrario, el estado de la señal de Q permanece en "0". En el parámetro PV se puede especificar una constante o una variable.
T3
El proceso de recuento se repite con cada flanco positivo hasta que el valor de recuento alcanza el valor máximo del tipo de dato definido en la salida CV. Una vez alcanzado este valor límite, el valor de contaje permanece constante, independientemente del estado de la señal en la entrada CU.
T4
Si hay un flanco positivo en la entrada R, el valor del contador en la salida CV se pone a "0".

A cada llamada de la instrucción "Count up" se le debe asignar una instancia del tipo de datos "CTU", en la que se almacenan los datos de la instancia.

Contador de conversión textual (CTU)
El contador ascendente CTU se instanciará de forma similar a un bloque de funciones. Para cada llamada al bloque de funciones "CTU" se necesita una zona de memoria independiente. Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

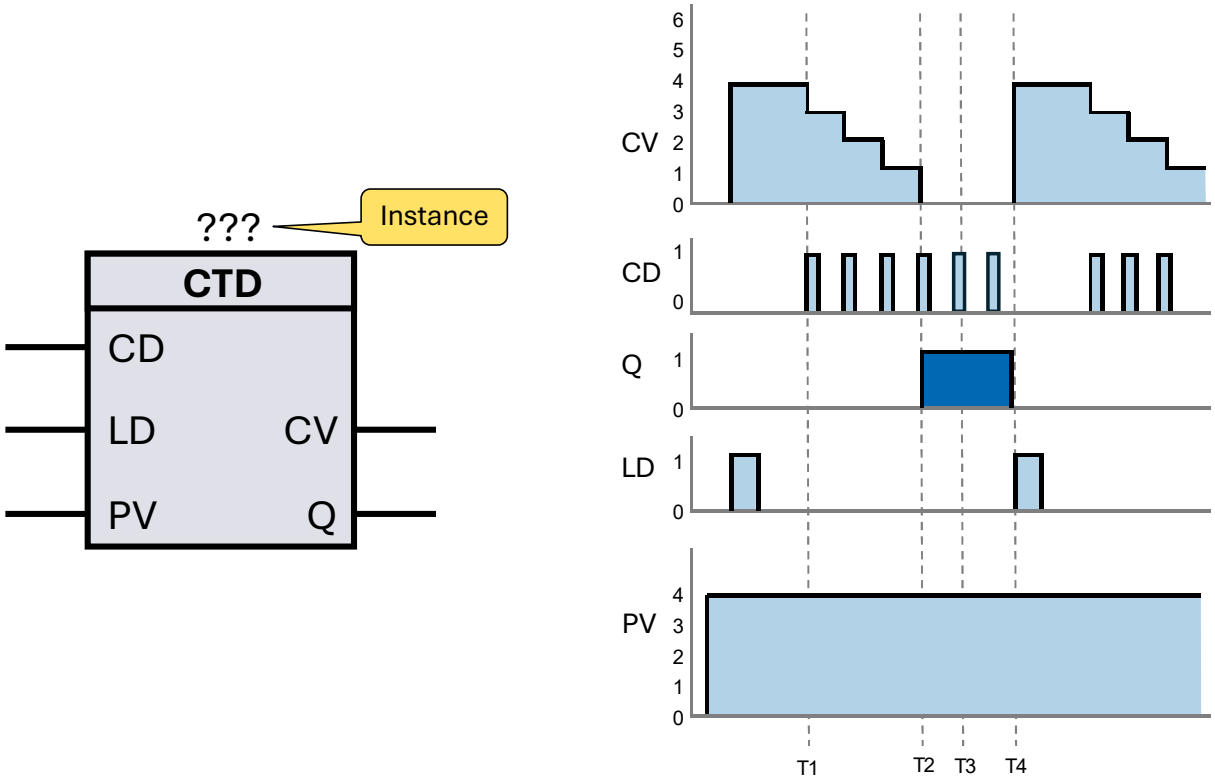
Por ejemplo:

```
//Declaration, Interface
VAR
   instCtu : CTU;   //Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instCtu(CU := varBoolCu,
        R := varBoolR,
        PV := varIntPv,
        CV => varIntCv,
        Q => varBoolQ);
```

Fotografía 26 Instrucción ST/SCL CTU

7.4.2 Cuenta atrás CTD

Puede programar un contador descendente con el contador IEC-CTD.
Con un flanco positivo en la entrada CD (Count Down), es decir, cuando el estado de la señal cambia de "0" a "1", el valor de contaje actual en la salida CV (Current Value) se reduce en uno. La salida Q (Salida) muestra el estado del contador. En cuanto el valor del contador es menor o igual que "0", la salida Q pasa a "1". Si el estado de la señal en la entrada LD (Carga) cambia de "0" a "1", el valor del contador en la salida CV se carga al valor del parámetro PV (Valor prefijado).



Fotografía 27 Instrucción FBD CTD y diagrama de impulsos

T1
Con un flanco positivo en la entrada CD, el valor de recuento actual en el valor de recuento CV se reduce en uno.
T2
Este proceso se repite con cada flanco positivo hasta que el valor de contaje CV alcanza el valor límite inferior del tipo de datos especificado. Si el valor de contaje actual CV es menor o igual que 0, la salida Q se pone a "1".
T3
Si se alcanza el valor límite inferior del tipo de datos especificado, la salida Q permanece en "1" y la entrada CD no tiene más influencia.
T4
Si hay un flanco positivo en la entrada LD, el valor especificado en la entrada PV se carga en la CV.

A cada llamada de la instrucción "Contar hacia atrás" se le debe asignar una instancia del tipo de datos "CTD", en la que se almacenan los datos de la instancia.

Contador de conversiones textuales (CTD)

El contador descendente CTD se instanciará de forma similar a un bloque de funciones. Se necesita una zona de memoria independiente para cada llamada del bloque de funciones "CTD". Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

Por ejemplo:

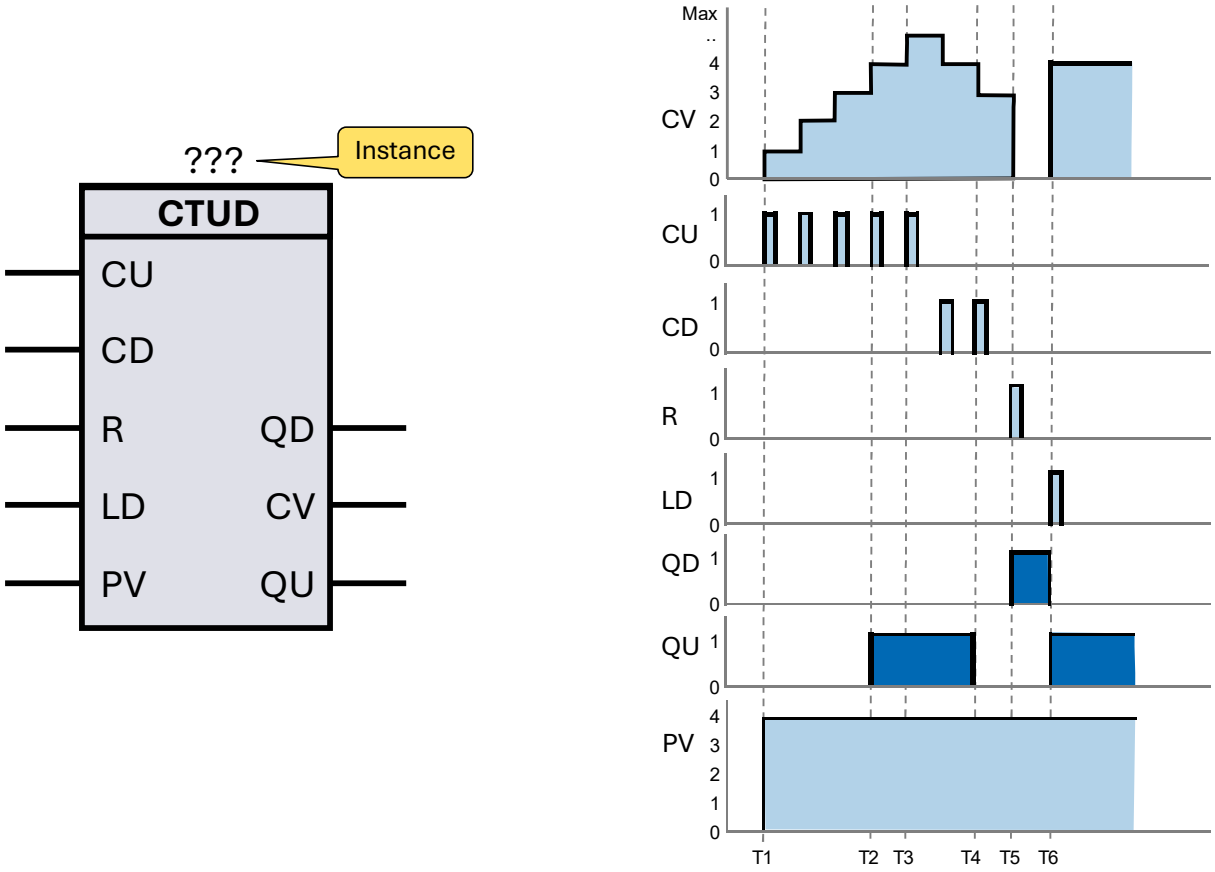
```
//Declaration, Interface
VAR
  instCtd : CTD;   //Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instCtd(CD := varBoolCd,
        LD := varBoolLd,
        PV := varIntPv,
        CV => varIntCv,
        Q => varBoolQ);
```

Fotografía 28 Instrucción ST/SCL CTD

7.4.3 Contar hacia delante y hacia atrás CTUD

Con la instrucción "Conteo ascendente y descendente", el valor de contaje en la salida CV (Valor actual) aumenta y disminuye. Un flanco positivo en la entrada CU (Count Up) aumenta el valor de la cuenta en uno, mientras que un flanco positivo en la entrada CD (Count Down) disminuye el valor de la cuenta en uno. Un flanco positivo en la entrada R (Reset) reinicia la salida CV a 0. Un flanco positivo en la entrada LD (Load) carga el valor especificado en la entrada PV (Preset Value) en el valor CV del contador. El contador tiene una salida QU (Count Up Output), que se activa si el valor de contaje actual CV es mayor o igual que el valor especificado en la entrada PV. El contador tiene una salida QD (Count Down Output), que se activa si el valor de contaje actual CV es menor o igual que 0.

Si en un ciclo de programa se produce un flanco de señal positivo en ambas entradas, el valor del contador permanece invariable.



Fotografía 29 Instrucción FBD CTUD y diagrama de impulsos

T1
Un flanco positivo en la entrada CU aumenta en uno el valor del contador CV.
T2
La salida QU se pone a "1" si el valor actual del contador CV es mayor o igual que el valor especificado en la entrada PV.
T3
El valor de recuento CV puede contarse hasta el valor límite superior o inferior del tipo de datos especificado.
T4

Cada flanco positivo en la entrada CD reduce el valor de recuento actual CV en uno. Si el valor de recuento actual CV es menor o igual que el valor especificado en la entrada PV, la salida QD se reinicia.

T5

Un flanco positivo en la entrada R pone a "0" el valor actual del contador, con lo que se activa la salida QD.

T6

Un flanco positivo en la entrada LD ajusta el valor actual del contador al valor especificado en la entrada PV, con lo que se activa la salida QD.

A cada llamada de la instrucción "Contar arriba y abajo" se le debe asignar una instancia del tipo de datos "CTUD", en la que se almacenan los datos de la instancia.

Conversión textual de contadores ascendentes y descendentes (CTUD)

El contador de subidas y bajadas CTUD se instancian de forma similar a un bloque de funciones. Para cada llamada al bloque de funciones "CTUD" se necesita una zona de memoria independiente. Los datos de instancia pueden crearse como instancia única o como multiinstancia (en la interfaz del bloque de funciones).

Ejemplo:

```
//Declaration, Interface
VAR
   instCtud : CTUD;   //Declaration of instance (Multi-instance)
END_VAR


//Instruction, instance call
instCtud(CU := varBoolCu,
         CD := varBoolCd,
         R := varBoolR,
         LD := varBoolLd,
         PV := varIntPv,
         QD => varBoolQd,
         CV => varIntCv,
         QU => varBoolQu);
```

Imagen 30 Instrucción ST/SCL CTUD

7.4.4 Ejercicio - Medidor IEC

Objetivo

En este ejercicio, se programan los 3 contadores IEC (CUT, CTD, CTUD) en un bloque de funciones de prueba y se familiariza con ellos en la práctica.
Los parámetros de entrada "countUp_signal" y "countDown_signal" pueden utilizarse para influir positiva o negativamente en el estado del recuento y ajustarse a valores definidos mediante los parámetros de entrada "reset_signal", "load_signal" y "presetValue". El estado se muestra a través de los parámetros de salida "output_CountUp", "output_CountDown", "output_Up_UpDown", "output_Down_UpDown".

Tarea

- Cree un bloque de funciones con la siguiente interfaz de bloque de funciones:

Name	Data type	Comment
Input		
countUp_signal	Bool	Test Input Count Up
countDown_signal	Bool	Test Input Countdown
reset_signal	Bool	Test Input Reset Counter
load_signal	Bool	Test Input Load
presetValue	Int	Test Input PV
Output		
output_CountUp	Bool	Test Output Q CTU
output_CountDown	Bool	Test Output QCTD
output_Up_UpDown	Bool	Test Output QU CTUD
output_Down_UpDown	Bool	Test Output QD CTUD
InOut		
<Add new>		
Static		
instCtu	CTU_INT	Instance CTU
instCtd	CTD_INT	Instance CTD
instCtud	CTUD_INT	Instance CTUD
Temp		

- En función del lenguaje de programación seleccionado, se aplicará el siguiente programa:


```
2  //Test CTU
3 #instCtu(CU := #countUp_signal,
4          R := #reset_signal,
5          PV := #presetValue,
6          //CV => ,
7          Q => #output_CountUp);
8
9  //Test CTD
10#instCtd(CD := #countDown_signal,
11         LD := #load_signal,
12         PV := #presetValue,
13         //CV=> ,
14         Q => #output_CountDown);
15
16 //Test CTUD
17#instCtud(CU := #countUp_signal,
18          CD := #countDown_signal,
19          R := #reset_signal,
20          LD := #load_signal,
21          PV := #presetValue,
22          //CV=> ,
23          QU => #output_Up_UpDown,
24          QD => #output_Down_UpDown);
```

- Llame el módulo creado en "MAIN".
- Las variables de entrada pueden controlarse y las de salida supervisarse a través de la instancia. Alternativamente, si están disponibles, también

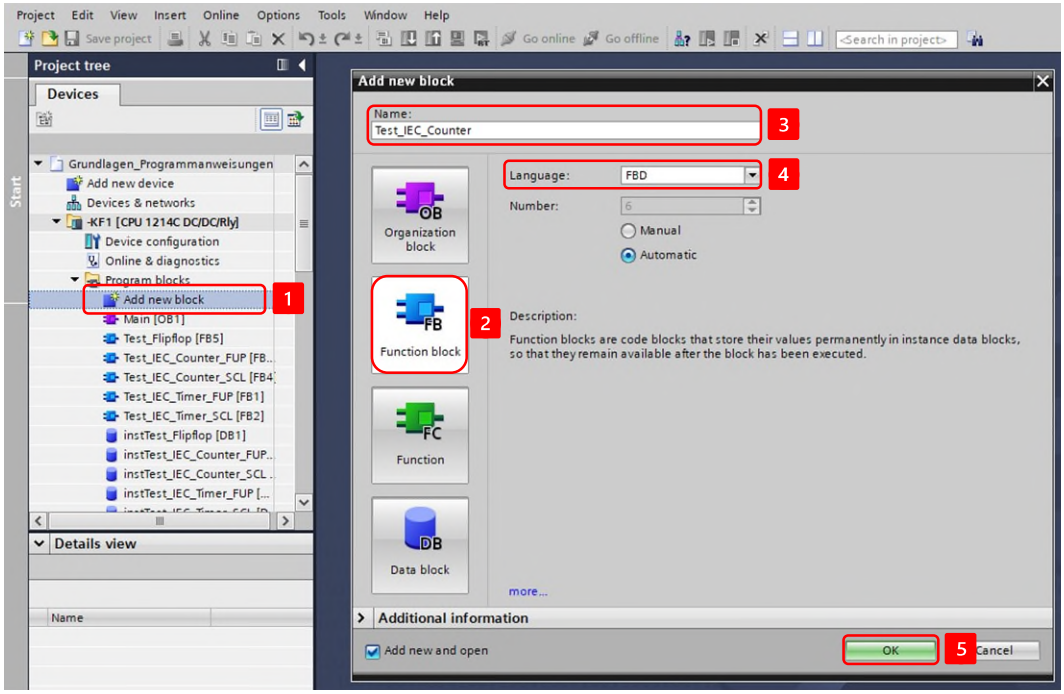
pueden conectarse a botones y elementos de visualización a través de la interfaz del bloque de funciones cuando se llama. El recuento también se puede supervisar en las instancias individuales de los contadores.

Como alternativa, también se pueden utilizar los bloques preparados "Test_IEC_Counter_FUP[FB3]" o "Test_IEC_Counter_SCL[FB4]" del proyecto de plantilla "Grundlagen_Programmanweisungen.zap17".

El capítulo "Puesta en servicio (software)" puede proporcionar ayuda adicional para interpretar el estado del programa.

Procedimiento:

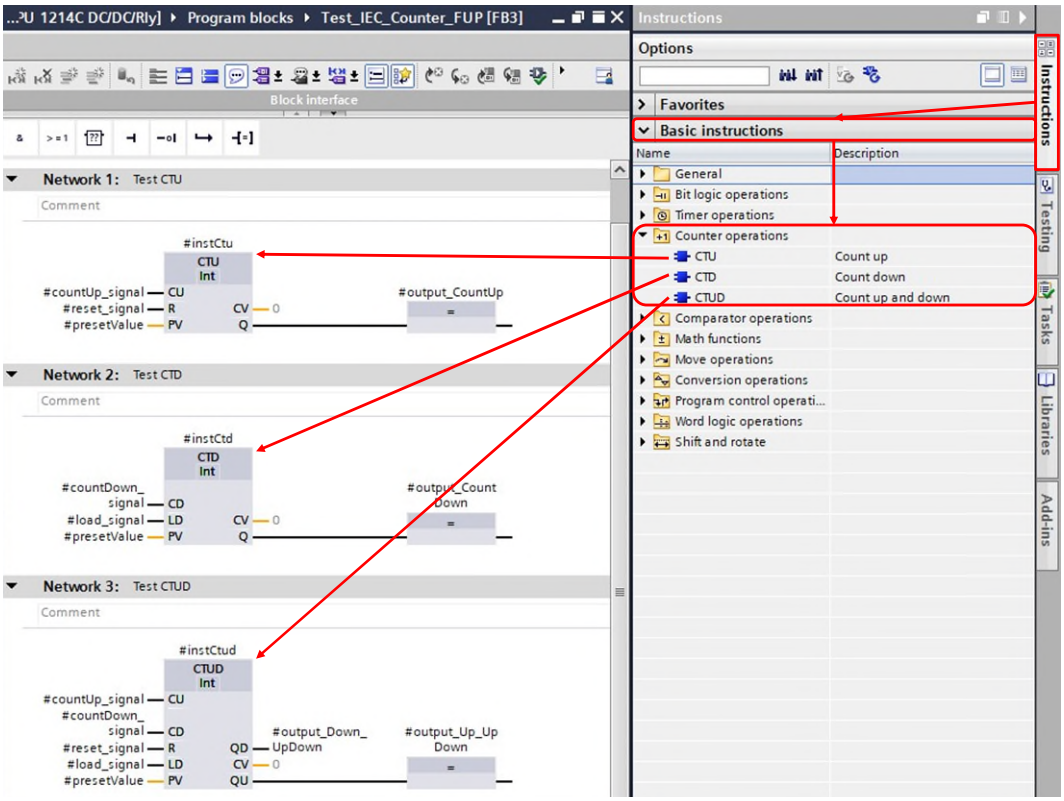
1. cree un nuevo bloque de funciones, seleccione el lenguaje de programación deseado y asígnele un nombre significativo:



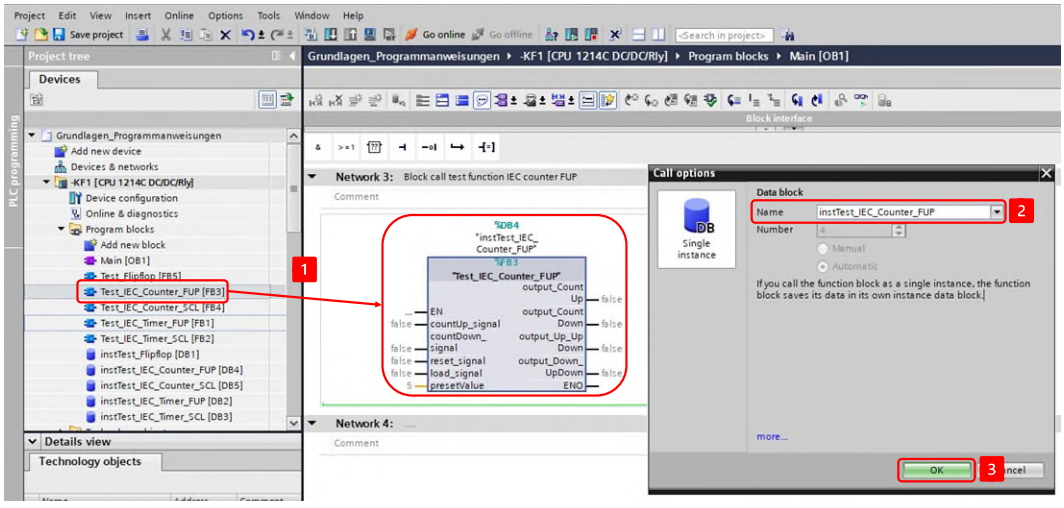
2. declare las siguientes variables en la interfaz del bloque de funciones:

	Name	Data type	Comment
1	Input		
2	countUp_signal	Bool	Test Input Count Up
3	countDown_signal	Bool	Test Input Countdown
4	reset_signal	Bool	Test Input Reset Counter
5	load_signal	Bool	Test Input Load
6	presetValue	Int	Test Input PV
7	Output		
8	output_CountUp	Bool	Test Output Q CTU
9	output_CountDown	Bool	Test Output QCTD
10	output_Up_UpDown	Bool	Test Output QU CTUD
11	output_Down_UpDown	Bool	Test Output QD CTUD
12	InOut		
13	<Add new>		
14	Static		
15	instCtu	CTU_INT	Instance CTU
16	instCtd	CTD_INT	Instance CTD
17	instCtud	CTUD_INT	Instance CTUD
18	Temp		

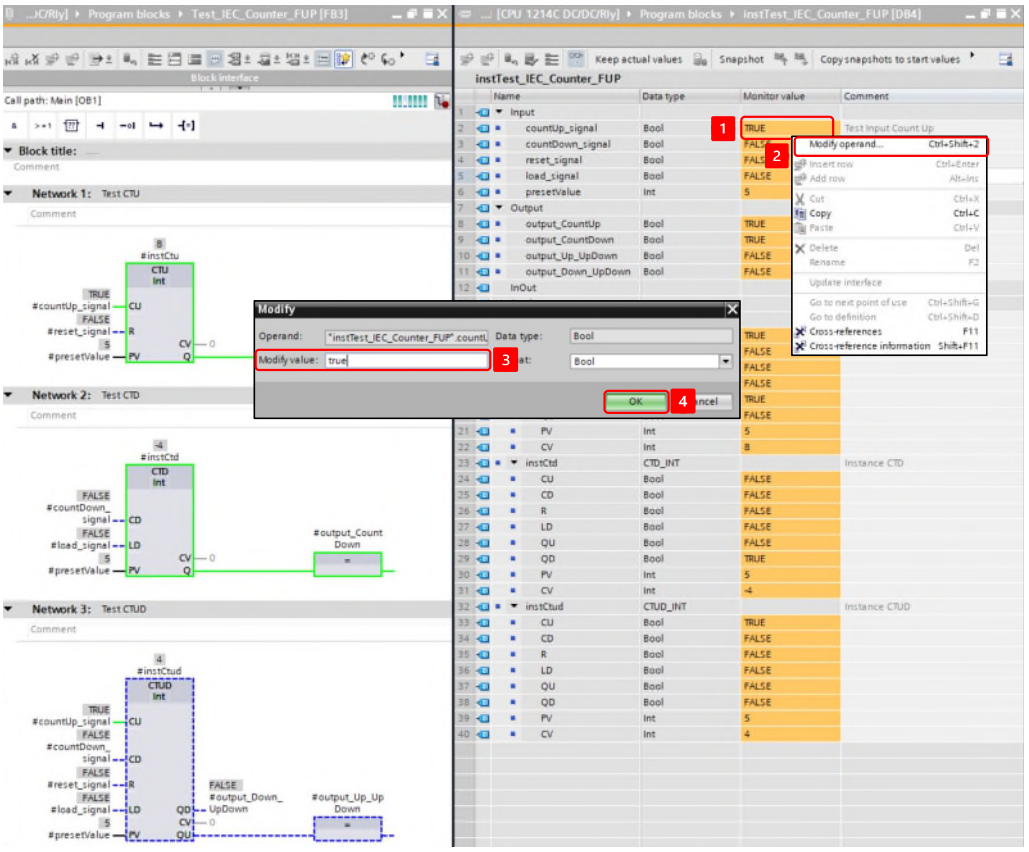
3. implemente el siguiente programa, las funciones de recuento se encuentran en la TaskCard en "Instrucciones"→ "Instrucciones simples"→ "Contador" :

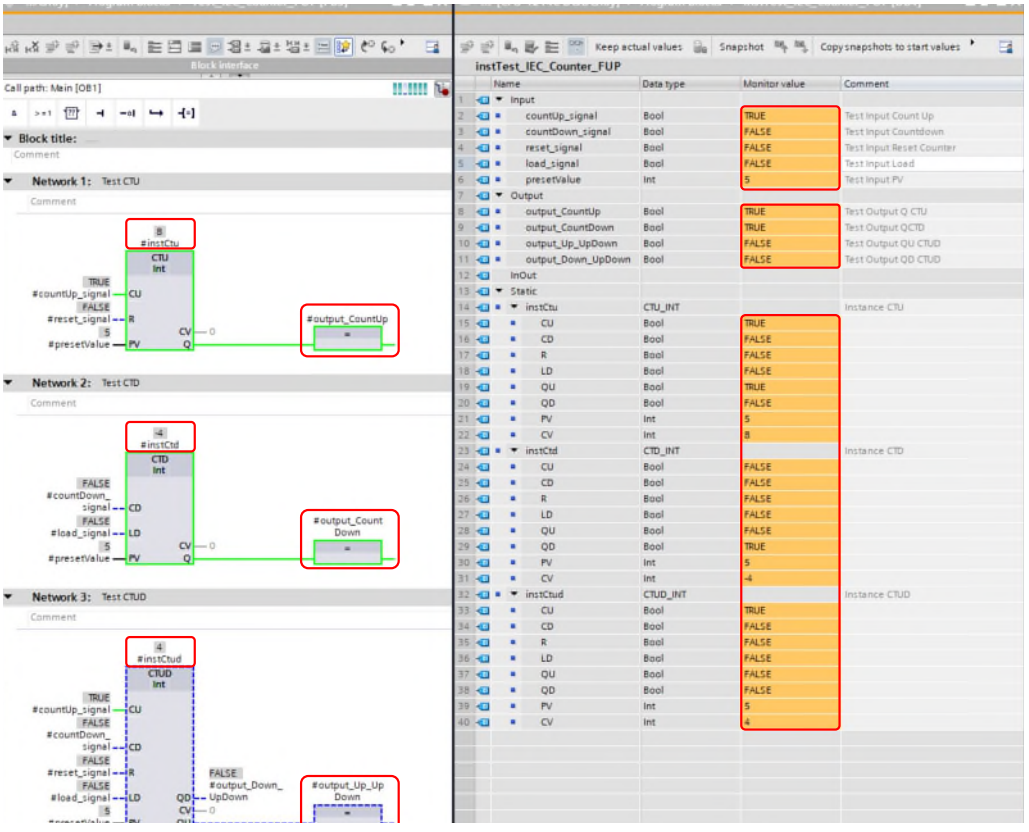


4. llame al bloque de funciones en "MAIN" y cree una instancia:



5. Utilice la instancia para controlar las variables de entrada y supervisar las variables de salida y las instancias individuales de las funciones de recuento en el bloque de datos de instancia correspondiente:





7.5 Declaración IF [ST / SCL]

En el lenguaje de programación ST / SCL, la sentencia IF se utiliza para controlar secuencias condicionales. Esta sentencia se puede utilizar para controlar el flujo del programa dependiendo de ciertas condiciones. A continuación se describe el uso de las sentencias IF, ELSE y ELSIF en SCL.

7.5.1 IF...THEN - Instrucción

La sentencia IF...THEN (Ejecutar condicionalmente) se utiliza para ejecutar un bloque de sentencia en función de una condición.

La estructura básica de esta instrucción es la siguiente:

```
IF (*condition*) THEN
    //statement

END_IF;
```

Fotografía 31 Ejemplo de código de la sentencia IF...THEN

Estructura del programa

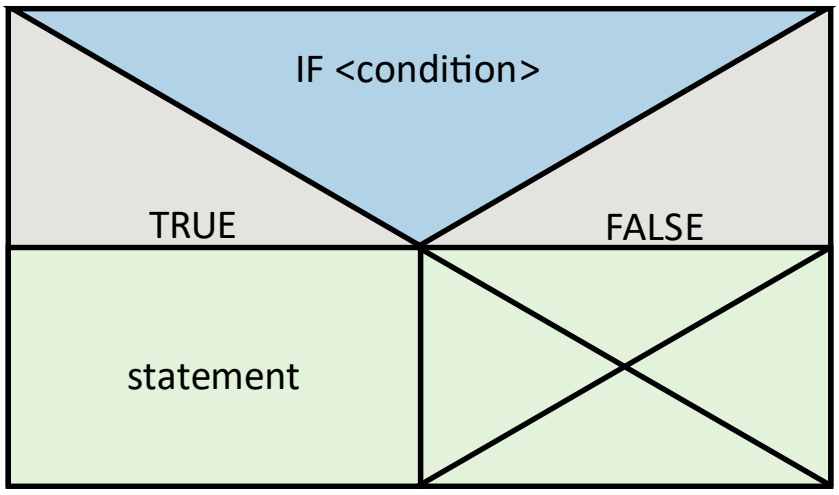


Imagen 32 Gráfico de estructura de la sentencia IF...THEN

Ejemplo

```
IF Temperatur > 100 THEN
    Alarm := TRUE;
END_IF;
```

Fotografía 33 Ejemplo de sentencia IF...THEN

En este ejemplo, la variable "Alarma" se pone a "TRUE" si se cumple la condición "Temperatura > 100".

7.5.2 Declaración IF...THEN...ELSE

La sentencia IF...THEN...ELSE (bifurcación condicional) se utiliza para bifurcar un bloque de sentencia en función de una condición.
La sentencia ELSE se utiliza para ejecutar sentencias alternativas si no se cumple la condición de la sentencia IF.

La estructura básica de esta instrucción es la siguiente:

```
IF (*condition*) THEN
    //statement 1

ELSE
    //statement 2

END_IF;
```

Imagen 34 Ejemplo de código de la sentencia IF...THEN...ELSE

Struktogram

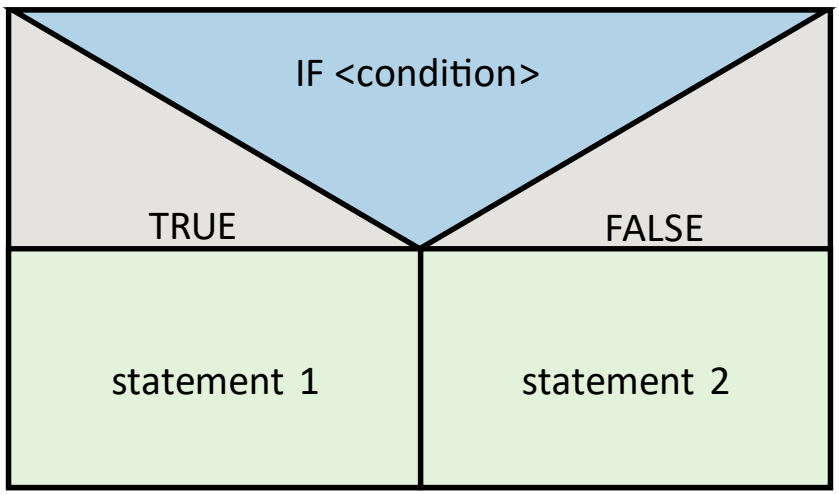


Imagen 35 Gráfico de estructura Declaración IF...THEN...ELSE

Ejemplo

```
IF Temperatur > 100 THEN
    Alarm := TRUE;
ELSE
    Alarm := FALSE;
END_IF;
```

Imagen 36 Ejemplo de sentencia IF...THEN...ELSE

En este ejemplo, la variable "Alarma" se pone a "TRUE" si se cumple la condición "Temperatura > 100". En caso contrario, "Alarma" se pone a "FALSE".

7.5.3 IF...THEN...ELSIF - Instrucción

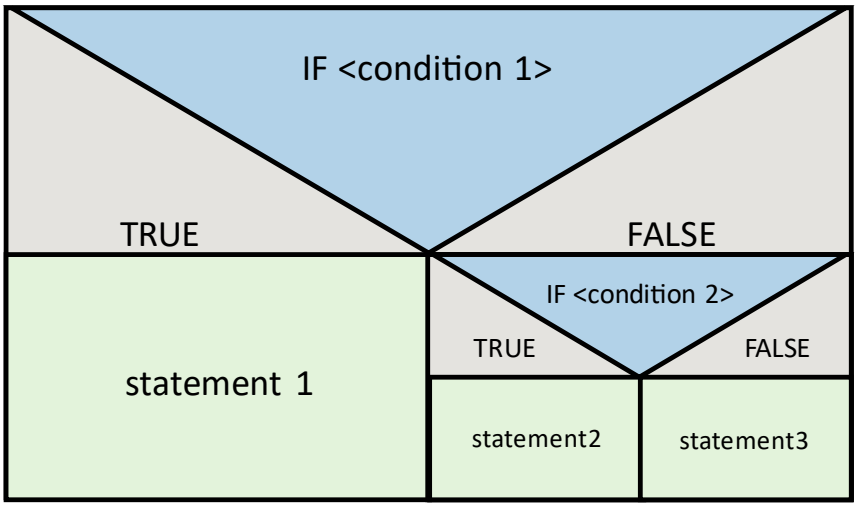
La sentencia IF...THEN...ELSIF (bifurcación condicional múltiple) se utiliza para bifurcar un bloque de sentencia en función de varias condiciones.
La sentencia ELSIF permite comprobar varias condiciones en una sola sentencia IF. Si no se cumple la primera condición, se comprueba la siguiente, y así sucesivamente.

La estructura básica de esta instrucción es la siguiente:

```
IF (*condition 1*) THEN
    //statement 1

ELSIF (*condition 2*) THEN
    //statement 2

ELSE
    //statement 3

END_IF;
```

Fotografía 37 Ejemplo de código de la sentencia IF...THEN...ELSIF

Estructura del programa



Fotografía 38 Gráfico de estructura Declaración IF...THEN...ELSE

Por ejemplo:

```
IF Temperatur > 100 THEN
    Alarm := TRUE;
ELSIF Temperatur > 80 THEN
    Warnung := TRUE;
ELSE
    Alarm := FALSE;
    Warnung := FALSE;
END_IF;
```

Imagen 39 Ejemplo de sentencia IF...THEN...ELSE

En este ejemplo:

- La variable "Alarma" se pone a "TRUE" si "Temperatura > 100".
- La variable "Advertencia" se pone a "VERDADERO" si la temperatura es superior a 80 pero inferior o igual a 100.
- De lo contrario, "Alarma" y "Advertencia" se ajustan a "FALSE".

7.6 Estructura CASE [ST / SCL]

Con la sentencia CASE (bifurcación múltiple / diferenciación de casos), se procesa una de varias secuencias de sentencias, en función del valor de una expresión numérica.
El valor de la expresión debe ser un número entero. Cuando se ejecuta la instrucción, el valor de la expresión se compara con los valores de varias constantes. Si el valor de la expresión coincide con el valor de una constante, se ejecutan las instrucciones programadas directamente después de esta constante.

Las constantes pueden asumir los siguientes valores:

- Un número entero (por ejemplo, 3)
- Un rango de números enteros (por ejemplo, 5...8)
- Una enumeración de números enteros y rangos (por ejemplo, 11, 17... 25)
- Secuencia de un bit (0001)

En función del valor de la expresión, se selecciona una de las siguientes alternativas (CASE1 ... CASEN) y se ejecuta la secuencia de sentencias correspondiente (sentencia 1 a sentencia N). Si no se aplica ninguna de las alternativas, se ejecuta la rama ELSE si está disponible.

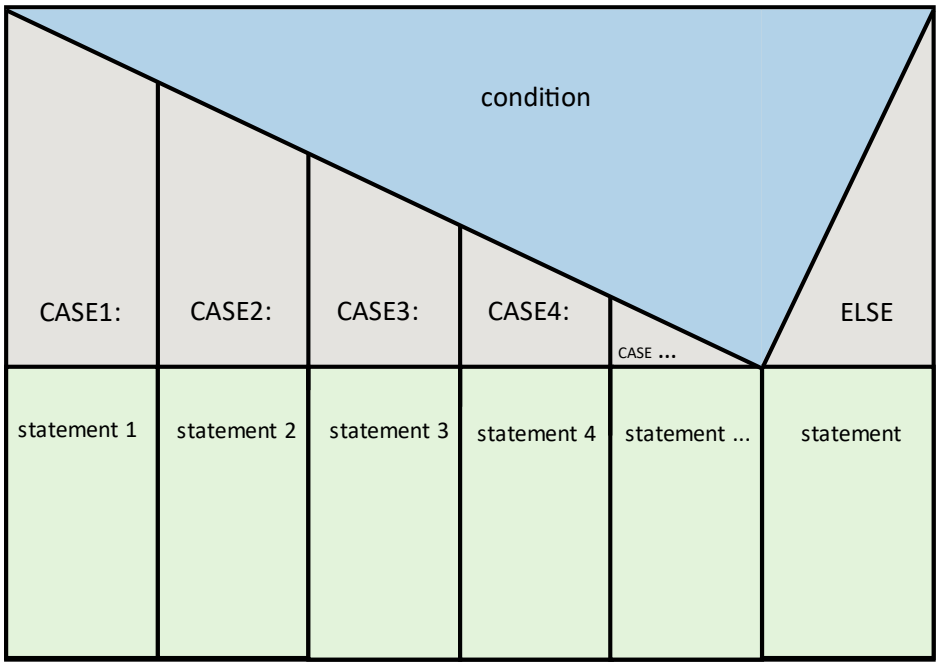


Imagen 40 Diagrama de la estructura de instrucciones CASE

La sintaxis de una instrucción CASE es la siguiente:

```
CASE (*condition*) OF
 1: //CASE 1
    //statement 1

 2: //CASE 2
    // statement 2

 3: //CASE 3
    // statement 3

 4: //CASE 4
    // statement 4

ELSE
    // statement

END_IF;
```

Fotografía 41 Sintaxis de la sentencia CASE

La estructura CASE también admite la especificación de rangos de valores y la agrupación de valores múltiples.

```
CASE (*Variable*) OF
 1..9: //Variable 1 - 9
    // statement

 10, 15, 19: //Variable 10, 15, 19
    // statement

ELSE
    // statement

END_IF;
```

Fotografía 42 Sentencia CASE con selección múltiple

En este ejemplo, la variable "Edad" está marcada y la categoría se asigna en función del intervalo de valores.

```
CASE age OF
 0..12: //Child
    Kategorie :='child';

 13..19: //Teenager
    Kategorie :='teenager';

 20..64: //Adult
     Kategorie :='adult';

 65..100: //Senior
     Kategorie :='senior';

ELSE
     Kategorie :='invalid';

END_IF;
```

Fotografía 43 Ejemplo CASO

La estructura CASE permite un método claro y eficaz para seleccionar entre varias alternativas en función del valor de una variable. Mejora la legibilidad y la facilidad de mantenimiento del código, sobre todo cuando hay que comprobar muchas condiciones, en comparación con la sentencia IF.
La opción de especificar rangos de valores y múltiples valores por caso ofrece una flexibilidad adicional.